Most Voip API Documentation

Release 0.1.0

Healthcare Flows - CRS4

Aug 08, 2017

Contents

The *MOST-Voip* Library is a fast and lightweight library created for handling VOIP sessions.

Main features:

- Sip Account creation and registration on a remote Sip Server (e.g Asterisk)
- Sip Call handling (making, holding, unholding, answering incoming calls)
- Buddies Subscription and Real Time Presence Notification

Supported platforms:

- Mobile: Android
- Desktop: Linux Ubuntu

So far, MOST-Voip for desktop platforms has been tested only on Linux Ubuntu v.14.04 distribution. However, it is written in Python 2.7, so other platforms should be supported as well.

CHAPTER 1

Table Of Contents

Python Most Voip Library

Contents:

Getting Started

The following tutorial shows you the main features of the library.

This tutorial assumes that you have installed and configured the Asterisk Sip Server on a reachable PC. (For getting instructions about the Asterisk manual configuration click on the Asterisk configuration link below)

Alternatively, you can download a virtual machine containing a running Asterisk Server instance already configured for running the proposed android examples, as explained here

Asterisk Configuration Guide for Most Voip Examples

All examples describing the Most Voip Library features require, to work properly, a Sip Server running on a reachable PC. In this guide we show how to configure the Asterisk Sip Server

Alternatively, if you prefer, you can install on your pc the Asterisk Virtual Machine (that contains an already configured Asterisk instance, as explained here.

How to add Sip Users to Asterisk

Open the **sip.conf** configuration file (generally located in the folder /etc/asterisk) set to **yes** the following options in the **[general]** section:

```
[general]
callevents=yes
notifyhold = yes
callcounter=yes
```

Also, add these sections at the end of ** sip.conf **:

```
[ste]
type=friend
secret=ste
host=dynamic
context=local_test

[steand]
type=friend
secret=steand
host=dynamic
context=local_test
```

How to add extensions to dial in Asterisk

Open the **extensions.conf** configuration file (generally located in the folder /etc/asterisk) and add these lines at the end:

```
[local_test]
exten => 1234,1,Answer ; answer the call
exten => 1234,2,Playback(tt-weasels) ; play an audio file that simulates the voice of␣
↪the called user
exten => 1234,3,Hangup ; hang up the call

exten => ste,1,Set(VOLUME(RX)=10) ; set the RX volume
exten => ste,2,Set(VOLUME(TX)=10) ; set the RX volume
exten => ste,hint,SIP/ste; hint  'ste' used for presence notification
exten => ste,3,Dial(SIP/ste) ; call the user ste'


exten => steand,1,Set(VOLUME(RX)=10) ; set the RX volume
exten => steand,2,Set(VOLUME(TX)=10) ; set the RX volume
exten => steand,hint,SIP/ste; hint  'steand' used for presence notification
exten => steand,3,Dial(SIP/steand) call the user 'steand' used for presence␣
↪notification
```

How to run Asterisk

Open a shell and type the following command:

```
sudo service asterisk restart
```

How to open the Asterisk Command Line Interface (CLI) Shell

```
sudo asterisk -r
```

How to look for sip users current state:

```
sip show peer
```

How to reload the dialplan (useful when you add and/or modify a new extension):

```
dialplan reload
```

How to originate a call:

This following command originates a call from the sip server to the user 'ste'. Obviously, it assumes that you have configured the Asterisk Server so that the user 'ste' is a known sip user. To do it , you have to configure the sip configuration file, called **sip.conf** (in Linux platforms, it is generally located in the folder /etc/asterisk).

```
originate SIP/ste extension
```

Tutorial 1: Making a Call

This first tutorial shows how to make a call to an arbitrary destination using the Voip Library. To make a call, you have to perform the following steps, each of them explained in the next sections.

Note that this example, to work, requires a Sip Server (e.g Asterisk) installed and running on a reachable PC. For getting instructions about the Asterisk configuration, click here

Step 1: Initialize the Library

First of all, you have to import and instance the class *VoipLib*

```
# add the most.voip library root dir to the current python path...
import sys
sys.path.append("../src/")

# import the Voip Library
from most.voip.api import VoipLib

# instanziate the lib
my_voip = VoipLib()
```

Now, you have to build a dictionary containing all parameters needed for the Lib initialization

```
# build a dictionary containing all parameters needed for the Lib initialization

voip_params = {  u'username': u'ste',  # a name describing the user
                 u'sip_server_address': u'192.168.1.100',  # the ip of the remote sip␣
↪server (default port: 5060)
                 u'sip_server_user': u'ste', # the username of the sip account
                 u'sip_server_pwd': u'ste',  #  the password of the sip account
                 u'sip_server_transport' :u'udp', # the transport type (default: tcp)
                 u'log_level' : 1,  # the log level (greater values provide more␣
↪informations)
```

```
                u'debug' : False  # enable/disable debugging messages
                }
```

At this point, you have to implement a callback method that will be

called by the voip library to notify any relevant voip event. You can choose an arbitrary name for this method, but it must contain the following 3 arguments: 1. *voip_event_type* argument indicating the type of the triggered event (VoipEventType.LIB_EVENT, VoipEventType.ACCOUNT_EVENT, VoipEventType.BUDDY_EVENT or VoipEventType.CALL_EVENT) | 2. *voip_event* reporting the specific event (e.g VoipEvent.ACCOUNT_REGISTERED to notify an account registration) 3. *params* a dictionary containing additional informations, depending on the specific triggered event call the *initialize* method passing the 2 parameters defined above

```
# define a method used for receive event notifications from the lib:

def notify_events(voip_event_type, voip_event, params):
    print "Received Event Type:%s -> Event: %s Params: %s" % (voip_event_type, voip_
↪event, params)
```

At this point, you are ready to initialize the library passing the dictionary and the callback method defined above:

```
# initialize the lib passing the dictionary and the callback method defined above:
my_voip.init_lib(voip_params, notify_events)
```

```
Received Event Type:EVENT_TYPE__LIB_EVENT -> Event: VOIP_EVENT__LIB_INITIALIZING␣
↪Params: {'params': {u'username': u'ste', u'sip_server_transport': u'udp', u'log_
↪level': 1, u'sip_server_user': u'ste', u'sip_server_pwd': u'ste', u'debug': False, u
↪'sip_server_address': u'192.168.1.100'}, 'success': True}
Received Event Type:EVENT_TYPE__LIB_EVENT -> Event: VOIP_EVENT__LIB_INITIALIZED␣
↪Params: {'sip_server': '192.168.1.100', 'success': True}
```

```
True
```

The example above assumes that you have a Sip Server (e.g, Asterisk) running on a pc reachable at the address 192.168.1.100.

Note that, so far, no connection to the Sip Server has been established yet. The *init_lib* method returns a *True* value if the initialization request completes without errors, *False* otherwise.

Finally, note that at the end of the inititialization process the method **notify_events** is called, containing all informations related to the outcome of the initialization process.

Step 2: Registering the account on the Sip Server

Now, you are ready to register the user to the sip server (in this example, we are registering a user called *ste* with the password *ste*. We assume that the Sip Server knows this user and is able to accept the registration request from it).

```
my_voip.register_account()
```

```
Received Event Type:EVENT_TYPE__ACCOUNT_EVENT -> Event: VOIP_EVENT__ACCOUNT_
↪REGISTERING Params: {'account_info': u'ste', 'Success': True}
```

```
True
```

Also in this case, the library calls the method *notify_events* to notify the outcome of the registration process. In particular, this method is called as soon as a registration request is sent (with a VoipEvent._ACCOUNT_REGISTERING event) and later, as soon as the registration is accepted by the remote Sip server (with a VoipEvent._ACCOUNT_REGISTERED state) or refused (with a VoipEvent._ACCOUNT_REGISTRATION_FAILED event)

Step 3: Making a call to an arbitrary extension

In case of successfull registration, you can dial an extension (or call an arbitrary Sip User) in the following way:

```
my_extension = "1234"
my_voip.make_call(my_extension)

import time
# wait until the call is active
while(True):
    time.sleep(1)
```

Note that the notify_events method is called when the call is established (with the event VoipEvent.CALL_ACTIVE)

Step 4: Hangup the active call

To hangup the call you have just to call the method *hangup_call*:

```
# ends the current call
my_voip.hangup_call()
```

```
True
```

Note that, when the user hangs up the call , the callback method is called again with the event VoipEvent.CALL_HANGUP)

Tutorial 2: Answering a Call

This second tutorial of the Most Voip Library shows how to listen for and to answer to incoming calls.

Note that this example, to work, requires a Sip Server (e.g Asterisk) installed and running on a reachable PC. For getting instructions about the Asterisk configuration, click here

The tutorial consists in the following steps, each of them explained in the next sections.

Step 1: Import and instance the voip lib

These steps have been already explained in the previous tutorial. However note that, this time, we also import the **VoipEvent** class, that will be used in the callback method **notify_events** for detecting the type of the incoming events.

```
# append the most voip library location to the pythonpath
import sys
sys.path.append("../src/")
```

```
# import the Voip Library
from most.voip.api import VoipLib
from most.voip.constants import VoipEvent
# instanziate the lib
my_voip = VoipLib()

# build a dictionary containing all parameters needed for the Lib initialization

voip_params = {  u'username': u'ste',  # a name describing the user
                 u'sip_server_address': u'192.168.1.100',  # the ip of the remote sip␣
↪server (default port: 5060)
                 u'sip_server_user': u'ste', # the username of the sip account
                 u'sip_server_pwd': u'ste',  #  the password of the sip account
                 u'sip_server_transport' :u'udp', # the transport type (default: tcp)
                 u'log_level' : 1,  # the log level (greater values provide more␣
↪informations)
                 u'debug' : False  # enable/disable debugging messages
                 }
```

Step 2: Implement the CallBack method where to receive notifications about incoming calls and other relevant events

```
import time
end_of_call = False # used as exit condition from the while loop at the end of this␣
↪example

# implement a method that will capture all the events triggered by the Voip Library
def notify_events(voip_event_type,voip_event, params):
    print "Received Event Type:%s  Event:%s -> Params: %s" % (voip_event_type, voip_
↪event, params)

    # event triggered when the account registration has been confirmed by the remote␣
↪Sip Server
    if (voip_event==VoipEvent.ACCOUNT_REGISTERED):
        print "Account %s registered: ready to accept call!" % myVoip.get_account().
↪get_uri()

    # event triggered when a new call is incoming
    elif (voip_event==VoipEvent.CALL_INCOMING):
        print "INCOMING CALL From %s" % params["from"]
        time.sleep(2)
        print "Answering..."
        myVoip.answer_call()

    # event triggered when the call has been established
    elif(voip_event==VoipEvent.CALL_ACTIVE):
        print "The call with %s has been established"  % myVoip.get_call().get_remote_
↪uri()

        dur = 4
        print "Waiting %s seconds before hanging up..."  % dur
        time.sleep(dur)
        myVoip.hangup_call()
```

```
    # events triggered when the call ends for some reasons
    elif (voip_event in [VoipEvent.CALL_REMOTE_DISCONNECTION_HANGUP, VoipEvent.CALL_
↪REMOTE_HANGUP, VoipEvent.CALL_HANGUP]):
        print "End of call. Destroying lib..."
        myVoip.destroy_lib()

    # event triggered when the library was destroyed
    elif (voip_event==VoipEvent.LIB_DEINITIALIZED):
        print "Call End. Exiting from the app."
        end_of_call = True

    # just print informations about other events triggered by the library
    else:
        print "Received unhandled event type:%s --> %s" % (voip_event_type,voip_event)
```

The method above detects the **VoipEvent.CALL_INCOMING** state, that is triggered when a remote user makes a call to the registered account (the user 'ste' in this example). In this example, we answer the incoming call and, in this way, the call is enstablished between the 2 users and the event **VoipEvent.CALL_CALLING** is triggered. At this point, we decide to wait 4 seconds before hanging up the call, by calling the **hangup_call** method. This method will end the current active call and will trigger the **VoipEvent.CALL_HANGUP** method (or one of the events **VoipEvent.CALL_REMOTE_DISCONNECTION_HANGUP** and **VoipEvent.CALL_REMOTE_HANGUP** if the remote user terminates the call before us), so we destroy the voip lib and wait for the **VoipEvent.LIB_DEINITIALIZED** event to set the flag **end_of_call** equals to True to notify the end of this example outside of this method.

Step 3: Initialize the Voip Library and register the account on the Sip Server

Now we have to initialize the library (by passing the notification method and the initialization params defined above) and register the account.

```
# initialize the lib passing the dictionary and the callback method defined above:
my_voip.init_lib(voip_params, notify_events)

# register the account
my_voip.register_account()
```

```
Received Event Type:EVENT_TYPE__LIB_EVENT  Event:VOIP_EVENT__LIB_INITIALIZING ->␣
↪Params: {'params': {u'username': u'ste', u'sip_server_transport': u'udp', u'log_
↪level': 1, u'sip_server_user': u'ste', u'sip_server_pwd': u'ste', u'debug': False, u
↪'sip_server_address': u'192.168.1.100'}, 'success': True}
Received unhandled event type:EVENT_TYPE__LIB_EVENT --> VOIP_EVENT__LIB_INITIALIZING
Received Event Type:EVENT_TYPE__LIB_EVENT  Event:VOIP_EVENT__LIB_INITIALIZED ->␣
↪Params: {'sip_server': '192.168.1.100', 'success': True}
Received unhandled event type:EVENT_TYPE__LIB_EVENT --> VOIP_EVENT__LIB_INITIALIZED
Received Event Type:EVENT_TYPE__ACCOUNT_EVENT  Event:VOIP_EVENT__ACCOUNT_REGISTERING -
↪> Params: {'account_info': u'ste', 'Success': True}
Received unhandled event type:EVENT_TYPE__ACCOUNT_EVENT --> VOIP_EVENT__ACCOUNT_
↪REGISTERING
```

```
True
```

Step 4: Add a 'while' loop for waiting for incoming calls

Now we are ready to wait for incoming call, so we can add a simple 'while loop' that doen't anything and exit when tha flag 'end_of_call' assumes the **true** value.

```
while (end_of_call==False):
    time.sleep(2)
```

Step 5: Originate a call from the Sip Server for testing the example

Open a CLI asterisk console and type the the following command for making a call to the user registered at the **step 3**:

originate SIP/ste extension

This commands originate a call from the sip server to the user 'ste' registered at the step 3. Obviously, it assumes that you have configured the Asterisk Server so that the user 'ste' is a known sip user. To do it , you have to configure the sip configuration file, called **sip.conf** (in Linux platforms, it is generally located in the folder /etc/asterisk).

; user section added at the end odf the configuration file sip.conf

[ste]

type=friend

secret=ste

host=dynamic

context=local_test

API

Most Voip Python API documentation.

Core Module

Most-Voip API - VoipLib Class

class most.voip.api.**VoipLib**(*backend=None*)

It is the core class of the Library, that allows you to:

- •initialize the Voip Library
- •create an account and register it on a remote Sip Server
- •make a call
- •listen for incoming calls and answer

answer_call()

Answer the current incoming call.

destroy_lib()

Destroy the Voip Lib and free all allocated resources.

get_account()

Get informations about the local account

Returns an *most.voip.interfaces.IAccount* object containing informations about the local sip account

get_call()
Get the current ICall instance

Returns an *most.voip.interfaces.ICall* object containing informations about the current call

get_server()
Get informations about the remote sip server

Returns an *most.voip.interfaces.IServer* object containing informations about the remote sip server

hangup_call()
Hangup the currently active call

hold_call()
Put the currently active call on hold status

init_lib(*params*, *notification_cb*)
Initialize the voip library

Parameters

- **params** – a dictionary containing all initialization parameters
- **notification_cb** – a callback method called by the library for all event notificationa (status changes, errors, events and so on)

Returns True if the initialization request completes without errors, False otherwise

make_call(*extension*)
Make a call to the specified extension

Parameters **extension** – the extension to dial

register_account()
Register the account specified into the *params* dictionary passed to the *init_lib()* method

unhold_call()
Put the currently active call on active status

unregister_account()
Unregister the account specified in the *params* dictionary passed to the *init_lib()* method

Core Interfaces

Most-Voip Interfaces

class most.voip.interfaces.**IAccount**
This class contains informations about the local sip account.

add_buddy(*extension*)
Add the specified buddy to this account (so its current state can be notified)

Parameters **extension** – the extension related to the buddy to add

get_buddies()
Get the list of buddies of the current registered account

Returns the list of *most.voip.interfaces.IBuddy* subscribed by the local account

get_buddy(*extension*)
Get the buddy with the given extension

Parameters **extension** – the extension of the buddy

Returns the *most.voip.interfaces.IBuddy* with the specified extension

get_state()

Returns the current state of this account (see *most.voip.constants.AccountState*)

get_uri()

Returns the sip uri of this account

remove_buddy(*extension*)
Remove the specified buddy from this account

Parameters **extension** – the extension related to the buddy to remove

class most.voip.interfaces.**IBuddy**
This class contains informations about a buddy. A buddy is a Sip user that notify its presence status to sip accounts that are interested to get informations by them.

get_extension()

Returns the sip extension of this buddy

get_state()

Returns the current state of this buddy (see *most.voip.constants.BuddyState*)

get_status_text()

Returns a textual description of the current status of this buddy

get_uri()

Returns the sip uri of this buddy

refresh_status()
Refreshes the current status of this buddy

class most.voip.interfaces.**ICall**
This class contains informations about a call between 2 sip accounts.

get_local_uri()

Returns the uri of the local sip account

get_remote_uri()

Returns the uri of the remote sip account

get_state()

Returns the current state of this call (see *most.voip.constants.CallState*)

class most.voip.interfaces.**IServer**
This class contains informations about the remote Sip Server (e.g Asterisk)

get_ip()

Returns the ip address of the remote sip server

get_state()

Returns the current status of the sip server (see *most.voip.constants.ServerState*)

Constants

Most-Voip Constants

class most.voip.constants.**VoipEvent**
This class contains all events triggered by the library

class most.voip.constants.**VoipEventType**
This class contains the list of different types of event triggerable by the library

ACCOUNT_EVENT = 'EVENT_TYPE__ACCOUNT_EVENT'
Account Event Type (account (un)registration)

BUDDY_EVENT = 'EVENT_TYPE__BUDDY_EVENT'
Buddy Event Type ((un)subsscribing, (dis)connection, remote (un)holding)

CALL_EVENT = 'EVENT_TYPE__CALL_EVENT'
Call Event Type (incoming, dialing, active, (un)holding, hanging up)

LIB_EVENT = 'EVENT_TYPE__LIB_EVENT'
Library Event Type (Library (de)initialization, Sip server (dis)onnection)

class most.voip.constants.**AccountState**
This class contains all allowed states of the local account

REGISTERED = 'SIP_ACCOUNT_STATE__REGISTERED'
Registered

UNREGISTERED = 'SIP_ACCOUNT_STATE__UNREGISTERED'
Unregistered

class most.voip.constants.**BuddyState**
This class contains all allowed states of a buddy

NOT_FOUND = 'BUDDY_STATE__NOT_FOUND'
Not Found

OFF_LINE = 'BUDDY_STATE__OFF_LINE'
Off line

ON_HOLD = 'BUDDY_STATE__ON_HOLD'
On hold

ON_LINE = 'BUDDY_STATE__ON_LINE'
On line

UNKNOWN = 'BUDDY_STATE__UNKNOWN'
Unknown

class most.voip.constants.**CallState**
This class contains all allowed states of a call

ACTIVE = 'CALL_STATE__ACTIVE'
Active call

DIALING = 'CALL_STATE__DIALING'
Dialing an outcoming call

HOLDING = 'CALL_STATE__HOLDING'
The local account put the active call on hold

IDLE = 'CALL_STATE__IDLE'
No call

INCOMING = 'CALL_STATE__INCOMING'
Dialing an incoming call

class most.voip.constants.**ServerState**
This class contains all allowed states of a remote Sip Server

CONNECTED = 'SIP_SERVER_STATE__CONNECTED'
Connected

DISCONNECTED = 'SIP_SERVER_STATE__DISCONNECTED'
Disconnected

NOT_FOUND = 'SIP_SERVER_STATE__NOT_FOUND'
Not Found

Examples

Basic usage examples can be found in the python tutorial page . Advanced examples can be found in the *python/examples/* subdirectory of the MOST-Voip sources.

Android Most Voip Library

Contents:

Getting Started

The following tutorial shows you the main features of the library on the Android platform.

This tutorial assumes that you have installed and configured the Asterisk Sip Server on a reachable PC. For getting instructions about the Asterisk configuration click here

Alternatively, you can download a virtual machine containing a running Asterisk Server instance already configured for running the proposed android examples, as explained here

[COMING SOON!]

Javadoc

most.voip.api

Utils

public class **Utils**

Methods

bytesToHex

public static String **bytesToHex**(byte[] *bytes*)
Convert byte array to hex string

Parameters

- **bytes** –

copyAssets

static void **copyAssets**(Context *ctx*)

getIPAddress

public static String **getIPAddress**(boolean *useIPv4*)

Get IP address from first non-localhost interface

Parameters

- **ipv4** – true=return ipv4, false=return ipv6

Returns address or empty string

getMACAddress

public static String **getMACAddress**(String *interfaceName*)

Returns MAC address of the given interface name.

Parameters

- **interfaceName** – eth0, wlan0 or NULL=use first interface

Returns mac address or empty string

getResourcePathByAssetCopy

public static String **getResourcePathByAssetCopy**(Context *ctx*, String *assetSubFolder*, String *fileToCopy*)

Copy the specified resource file from the assets folder into the "files dir" of this application, so that this resource can be opened by the Voip Lib by providing it the absolute path of the copied resource

Parameters

- **ctx** – The application context
- **assetPath** – The path of the resource (e.g on_hold.wav or sounds/on_hold.wav)

Returns the absolute path of the copied resource, or null if no file was copied.

getUTF8Bytes

public static byte[] **getUTF8Bytes**(String *str*)

Get utf8 byte array.

Parameters

- **str** –

Returns array of NULL if error was found

loadFileAsString

public static String **loadFileAsString**(String *filename*)
Load UTF8withBOM or any ansi text file.

Parameters

- **filename** –

Throws

- **java.io.IOException** –

VoipEventBundle

public class **VoipEventBundle**

Constructors

VoipEventBundle

public **VoipEventBundle**(*VoipEventType eventType*, *VoipEvent event*, String *info*, Object *data*)
This object contains all the informations of a Sip Event triggered by the Voip Library

Parameters

- **eventType** – the type of this event
- **event** – the event
- **info** – a textual information describing this event
- **data** – a generic object containing event-specific informations (the object type depends on the type of the event)

Methods

getData

public Object **getData**()
Get a generic object containing event-specific informations (the object type depends on the type of the event)

Returns a generic object containing event-specific informations

getEvent

public *VoipEvent* **getEvent**()

getEventType

public *VoipEventType* **getEventType**()
Get the event type

Returns the event type

getInfo

public String **getInfo**()
Get a textual description of this event

Returns a textual description of this event

VoipLib

public interface **VoipLib**
It is the core class of the Library, that allows you to:

•initialize the Voip Library

•create an account and register it on a remote Sip Server

•make a call

•listen for incoming calls and answer

•get instances of IAccount, ICall and IServer objects

Methods

answerCall

public boolean **answerCall**()
Answer a call

Returns false if this command was ignored for some reasons (e.g there is already an active call), true otherwise

destroyLib

public boolean **destroyLib**()
Destroy the Voip Lib

Returns `true` if no error occurred in the deinitialization process

getAccount

public *IAccount* **getAccount**()
Get informations about the local sip account

Returns informations about the local sip account , like its current state

getCall

public *ICall* **getCall**()
Get the current call info (if any)

Returns informations about the current call (if any), like the current Call State

getServer

public *IServer* **getServer**()
Get informations about the remote Sip Server

Returns informations about the current sip server, like the current Server State

hangupCall

public boolean **hangupCall**()
Close the current active call

Returns true if no error occurred during this operation, false otherwise

holdCall

public boolean **holdCall**()
Put the active call on hold status

Returns true if no error occurred during this operation, false otherwise

initLib

public boolean **initLib**(Context *context*, HashMap<String, String> *configParams*, Handler *notificationHandler*)
Initialize the Voip Lib

Parameters

- **context** – application context of the activity that uses this library
- **configParams** – All needed configuration string parameters. All the supported parameters are the following (turn server params are needed only if you intend to use a turn server):

•sipServerIp: the ip address of the Sip Server (e.g Asterisk)

•sipServerPort: the port of the Sip Server (default: 5060)

•sipServerTransport: the sip transport: it can be “udp” or “tcp” (default: “udp”)

•sipUserName: the account name of the peer to register to the sip server

•sipUserPwd: the account password of the peer to register to the sip server

•turnServerIp: the ip address of the Turn Server

•turnServerPort: the port of the Turn Server (default: 3478)

•turnServerUser: the username used for TurnServer Authentication

•turnServerPwd: the password of the user used for TurnServer Authentication

•turnAuthRealm: the realm for the authentication (default:”most.crs4.it”)

•onHoldSound: the path of the sound file played when the call is put on hold status

•onIncomingCallSound: the path of the sound file played for outcoming calls

•onOutcomingCallSound: the path of the sound file played for outcoming calls

Parameters

- **notificationHandler** – the handller that will receive all sip notifications

Returns true if the initialization request completes without errors, false otherwise

makeCall

public boolean **makeCall** (String *extension*)
Make a call to the specific extension

Parameters

- **extension** – The extension to dial

Returns true if no error occurred during this operation, false otherwise

registerAccount

public boolean **registerAccount** ()
Register the account according to the configuration params provided in the *initLib(HashMap,Handler)* method

Returns `true` if the registration request was sent to the sip server, `false` otherwise

unholdCall

public boolean **unholdCall** ()
Put the active call on active status

Returns true if no error occurred during this operation, false otherwise

unregisterAccount

public boolean **unregisterAccount** ()
Unregister the currently registered account

Returns `true` if the unregistration request was sent to the sip server, `false` otherwise

VoipLibBackend

public class **VoipLibBackend** extends Application implements *VoipLib*
This class implements the *most.voip.api.VoipLib* interface by using the PJSip library as backend. So, you can get a *most.voip.api.VoipLib* instance in the following way:

```
VoipLib myVoip = new VoipLibBackend();
```

To get a *most.voip.api.interfaces.ICall* instance you can call the *getCall()* method:

```
ICall myCall = myVoip.getCall();
```

To get a *most.voip.api.interfaces.IAccount* instance you can call the *getAccount()* method:

```
IAccount myAccount = myVoip.getAccount();
```

To get a *most.voip.api.interfaces.IServer* instance you can call the *getServer()* method:

```
IServer mySipSever = myVoip.getServer();
```

See also: *VoipLib*

Constructors

VoipLibBackend

public **VoipLibBackend**()

Methods

answerCall

public boolean **answerCall**()

destroyLib

public boolean **destroyLib**()

getAccount

public *IAccount* **getAccount**()

getCall

public *ICall* **getCall**()

getServer

public *IServer* **getServer**()

getSipUriFromExtension

public String **getSipUriFromExtension**(String *extension*)

Get a sip uri in the format sip:@sip_server_ip[:sip_server_port]

Parameters

- **extension** – the extension of the sip uri

Returns the sip uri

hangupCall

public boolean **hangupCall** ()

holdCall

public boolean **holdCall** ()

initLib

public boolean **initLib** (Context *context*, HashMap<String, String> *configParams*, Handler *notificationHandler*)

makeCall

public boolean **makeCall** (String *extension*)

registerAccount

public boolean **registerAccount** ()

unholdCall

public boolean **unholdCall** ()

unregisterAccount

public boolean **unregisterAccount** ()

most.voip.api.enums

AccountState

public enum **AccountState**

Enum Constants

REGISTERED

public static final *AccountState* **REGISTERED**

UNREGISTERED

public static final *AccountState* **UNREGISTERED**

BuddyState

public enum **BuddyState**

Enum Constants

NOT_FOUND

public static final *BuddyState* **NOT_FOUND**

OFF_LINE

public static final *BuddyState* **OFF_LINE**

ON_HOLD

public static final *BuddyState* **ON_HOLD**

ON_LINE

public static final *BuddyState* **ON_LINE**

UNKNOWN

public static final *BuddyState* **UNKNOWN**

CallState

public enum **CallState**

Enum Constants

ACTIVE

public static final *CallState* **ACTIVE**
The call is active

DIALING

public static final *CallState* **DIALING**
An outcoming call is ringing

HOLDING

public static final *CallState* **HOLDING**
The call is on hold state

IDLE

public static final *CallState* **IDLE**
No call

INCOMING

public static final *CallState* **INCOMING**
The incoming call is ringing

RegistrationState

public enum **RegistrationState**

Enum Constants

FORBIDDEN

public static final *RegistrationState* **FORBIDDEN**

NOT_FOUND

public static final *RegistrationState* **NOT_FOUND**

OK

public static final *RegistrationState* **OK**

REQUEST_TIMEOUT

public static final *RegistrationState* **REQUEST_TIMEOUT**

SERVICE_UNAVAILABLE

public static final *RegistrationState* **SERVICE_UNAVAILABLE**

ServerState

public enum **ServerState**

Enum Constants

CONNECTED

public static final *ServerState* **CONNECTED**

DISCONNECTED

public static final *ServerState* **DISCONNECTED**

VoipEvent

public enum **VoipEvent**
Contains all events triggered by the library

Enum Constants

ACCOUNT_REGISTERED

public static final *VoipEvent* **ACCOUNT_REGISTERED**
The sip user has been successfully registered to the remote Sip Server (this event is also triggered called for each registration renewal)

ACCOUNT_REGISTERING

public static final *VoipEvent* **ACCOUNT_REGISTERING**
The Sip user is under registration process (this event triggered only for explicit registration requests, so it is no called during automatic registration renewals)

ACCOUNT_REGISTRATION_FAILED

public static final *VoipEvent* **ACCOUNT_REGISTRATION_FAILED**
The User Account Registration process failed for some reason (e.g authentication failed)

ACCOUNT_UNREGISTERED

public static final *VoipEvent* **ACCOUNT_UNREGISTERED**
The sip user has been successfully unregistered

ACCOUNT_UNREGISTERING

public static final *VoipEvent* **ACCOUNT_UNREGISTERING**
The Sip user is under unregistration process

ACCOUNT_UNREGISTRATION_FAILED

public static final *VoipEvent* **ACCOUNT_UNREGISTRATION_FAILED**
The User Account Unregistration process failed for some reason (e.g the sip server is down)

BUDDY_CONNECTED

public static final *VoipEvent* **BUDDY_CONNECTED**
The remote user is connected (i.e is in ON LINE status)

BUDDY_DISCONNECTED

public static final *VoipEvent* **BUDDY_DISCONNECTED**
The remote user is no longer connected (i.e is in OFF LINE status)

BUDDY_HOLDING

public static final *VoipEvent* **BUDDY_HOLDING**
The remote user is still connected, but it is not available at the moment (it is in BUSY state)

BUDDY_SUBSCRIBED

public static final *VoipEvent* **BUDDY_SUBSCRIBED**
The remote user has been successfully subscribed (it is now possible to get status notifications about it)

BUDDY_SUBSCRIBING

public static final *VoipEvent* **BUDDY_SUBSCRIBING**
a remote user is under subscrition process

BUDDY_SUBSCRIPTION_FAILED

public static final *VoipEvent* **BUDDY_SUBSCRIPTION_FAILED**
The remote user subscription process failed for some reason

CALL_ACTIVE

public static final *VoipEvent* **CALL_ACTIVE**
The call is active

CALL_DIALING

public static final *VoipEvent* **CALL_DIALING**
an outcoming call is ringing

CALL_HANGUP

public static final *VoipEvent* **CALL_HANGUP**
The local user hangs up

CALL_HOLDING

public static final *VoipEvent* **CALL_HOLDING**
The local user puts on hold the call

CALL_INCOMING

public static final *VoipEvent* **CALL_INCOMING**
an incoming call is ringing

CALL_READY

public static final *VoipEvent* **CALL_READY**
a new call is ready to become active or rejected

CALL_REMOTE_DISCONNECTION_HANGUP

public static final *VoipEvent* **CALL_REMOTE_DISCONNECTION_HANGUP**
The remote server has been disconnected so the call was interrupted.

CALL_REMOTE_HANGUP

public static final *VoipEvent* **CALL_REMOTE_HANGUP**
The remote user hangs up

CALL_UNHOLDING

public static final *VoipEvent* **CALL_UNHOLDING**
The local user unholds the call

LIB_CONNECTION_FAILED

public static final *VoipEvent* **LIB_CONNECTION_FAILED**
The connection to the remote Sip Server failed (a Timeout occurred during account an registration request tothe remote Sip Server)

LIB_DEINITIALIZATION_FAILED

public static final *VoipEvent* **LIB_DEINITIALIZATION_FAILED**
The library deinitialization process failed for some reason (e.g authentication failed)

LIB_DEINITIALIZED

public static final *VoipEvent* **LIB_DEINITIALIZED**
The lib was successully deinitialied

LIB_DEINITIALIZING

public static final *VoipEvent* **LIB_DEINITIALIZING**
The library is under deinitilization process

LIB_INITIALIZATION_FAILED

public static final *VoipEvent* **LIB_INITIALIZATION_FAILED**
The library initialization process failed for some reason (e.g authentication failed)

LIB_INITIALIZED

public static final *VoipEvent* **LIB_INITIALIZED**
The lib was successully initialied

LIB_INITIALIZING

public static final *VoipEvent* **LIB_INITIALIZING**
The library is under initilization process

VoipEventType

public enum **VoipEventType**

Enum Constants

ACCOUNT_EVENT

public static final *VoipEventType* **ACCOUNT_EVENT**
Voip Account Events ((un)registration)

BUDDY_EVENT

public static final *VoipEventType* **BUDDY_EVENT**
Voip Buddy Events (buddy presence notification: (un)subsscribing, (dis)connection, remote (un)holding)

CALL_EVENT

public static final *VoipEventType* **CALL_EVENT**
Voip Call Events (incoming, dialing, active, (un)holding, hanging up)

LIB_EVENT

public static final *VoipEventType* **LIB_EVENT**
Voip Library Events (Voip (de)initialization)

most.voip.api.interfaces

IAccount

public interface **IAccount**
Represents a local sip account

Methods

addBuddy

public boolean **addBuddy**(String *uri*)
Add a buddy to this account.

Parameters

- **uri** – the buddy sip uri

Returns True if the buddy was added to the buddy list, False otherwise

getBuddies

public *IBuddy*[] **getBuddies**()
Get the list of buddies of the current registered account

Returns the list of the buddies of the currently registered account

getBuddy

public *IBuddy* **getBuddy**(String *uri*)
Get the buddy with the given extension, or null if it is not found

Parameters

- **uri** – the buddy uri

Returns the buddy with the provided uri, or null if it is not found

getState

public *AccountState* **getState**()
Get the current state of this account

Returns the current state of this account

getUri

public String **getUri**()
Get the uri of this sip account

Returns the sip uri of this account

removeBuddy

public boolean **removeBuddy**(String *uri*)
Remove the buddy from this account

Parameters

- **uri** – The sip uri of the buddy to remove

Returns True if the buddy was found and it was successfully removed, False otherwise

IBuddy

public interface **IBuddy**
An IBuddy is a remote Sip user that notify its presence status to sip accounts (*IAccount* objects) that are interested to get informations by them.

Methods

getExtension

String **getExtension**()
get the sip extension of this buddy

Returns the sip extension of this buddy

getState

BuddyState **getState**()
get the current state of this buddy

Returns the current state of this buddy

See also: *IBuddy.refreshStatus()*

getStatusText

String **getStatusText**()
get a textual description of the current status of this buddy

Returns a textual description of the current status of this buddy

getUri

String **getUri**()
get the sip uri of this buddy

Returns the sip uri of this buddy

refreshStatus

void **refreshStatus**()
Refreshes the current status of this buddy

ICall

public interface **ICall**
Contains informations about a call between 2 sip accounts.

Methods

getLocalUri

String **getLocalUri**()
get the uri of the local sip account

Returns the uri of the local sip account

getRemoteUri

String **getRemoteUri**()
get the uri of the remote sip account

Returns the uri of the remote sip account

getState

CallState **getState**()
get the current state of this call

Returns the current state of this call

IServer

public interface **IServer**
Contains informations about the remote Sip Server (e.g Asterisk)

Methods

getIp

String **getIp**()
get the ip address of the remote sip server

Returns the ip address of the remote sip server

getPort

String **getPort**()
get the port of the remote sip server

Returns the ip address of the remote sip server

getState

ServerState **getState**()
get the current status of the sip server (see *most.voip.constants.ServerState*)

Returns the current status of the sip server

Examples

A tutorial can be found in the getting started page . Basic and advanced examples can be found in the *android/examples/* subdirectory of the MOST-Voip sources. The available examples are the following:

- *MostVoipActivityFirstExample*: shows how to initialize the Voip Lib and register a Sip Account on a remote Sip Server
- *MostVoipActivitySecondExample*: shows how to make a call to a remote Sip account
- *MostVoipActivityAnswerCallExample*: shows how to answer a call incoming from a remote Sip account
- *MostVoipActivityCallStateExample*: shows how to monitor the state of the remote Sip Server, of the current call and of the remote buddies
- *MostVoipActivityDemo*: show how to make a call to a remote buddy, answer a call, and monitor the state of the remote Sip Server, of the current call and of the remote buddies
- *MostVoipActivityRemoteConfigurationExample*: like the previous example, but it also shows how to load the Sip Account Configuration from a remote Web Server

How to build and run the examples

First of all, download the Most-voip Asterisk VM, containing a running Asterisk Server instance already configured for running the proposed android examples, as explained here .

Then, do the following:

- Open your preferred IDE and import the Android Most Voip library project from the *android/src/AndroidVoipLib* folder (if you are using Eclipse, select *File/Import.../Android/Existing Android Code Into Workspace* to import the project)

- Add to the project the dependence *android-support-v4.jar* . Please, visit this site to get detailed instructions about how to do it.
- Import your preferred example project (e.g MostVoipActivityFirstExample) located in the *android/examples* folder in the same way you have imported the Android Most Voip library project
- Set the AndroidVoipLib library project (previously added to the workspace) as a Project Reference of the example project imported at the previous step
- Add to the example project the dependence 'android-support-v4.jar' , in the same way you have done for the AndroidVoipLib library project
- Build the example project and deploy the generated .apk on your android emulator or mobile phone

Authors

Code author: Francesco Cabras <francesco.cabras@crs4.it>

Code author: Stefano Leone Monni <stefano.monni@crs4.it>

CHAPTER 2

Installation

Most-Voip Library is based on PJSIP 2.2.1 library. So, first of all, you have to install PJSip, by performing the following steps:

1. Download the last svn revision from http://svn.pjsip.org/repos/pjproject/trunk/ (revision 4818 works well). (tar.gz and zip archives don't compile!)
2. ./configure CFLAGS='-fPIC'
3. make dep
4. make
5. sudo make install
6. cd pjsip-apps/src/python/
7. sudo python setup.py install

If you intend to use Most-Voip on the Android platform, you also have to build Pjsip for Android, as explained here

Get the latest release from GitHub: https://github.com/crs4/most-voip

CHAPTER 3

License

```
Project MOST - Moving Outcomes to Standard Telemedicine Practice
http://most.crs4.it/

Copyright 2014, CRS4 srl. (http://www.crs4.it/)

Dual licensed under the MIT or GPL Version 2 licenses.
See license-GPLv2.txt or license-MIT.txt
```

GPL2: https://www.gnu.org/licenses/gpl-2.0.txt

MIT: http://opensource.org/licenses/MIT

CHAPTER 4

Detailed Dual Licensing Info

The MOST-Voip API is licensed under both General Public License (GPL) version 2 and the MIT licence. In practical sense, this means:

- if you are developing Open Source Software (OSS) based on MOST-Voip, chances are you will be able to use MOST-Voip under GPL. Note that the Most-Voip Library depends on the PJSIP API, so please double check here for OSS license compatibility with GPL.
- alternatively, you can release your application under MIT licence, provided that you have followed the guidelines of the PJSIP licence explained here.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

H

I

L

M

N

O

R

S

U

V