聯陽半導體
ITE Tech. Inc.

iTE 可視對講系統

Software Developer's Manual

Android Mobile Devices:

SIP Phone Programming Guide

V0.9

ITE TECH. INC

DATE	DESCRIPTION	PAGE
2015/9/23	Create the document	

1. FOREWORD

This document elucidates how to initialize and use the Linphone SDK to implement some commonly used functionality, such as audio and video calls, in your Android app. Besides, this document also talks a little bit about the iTE-proprietary network protocol, for example, open the door.

1.1 PREREQUISITES

Experience in Object Oriented Programming
Experience in Java Language Programming
Experience in JNI Programming
Experience in Android Application Development

1.2 INTENTION

This document intends to facilitate developers to pick up the core knowledge about how to write a SIP-based mobile app by using the Linphone SDK on the Android system.

1.3 AUDIENCE

Software Developers
Software Integrators

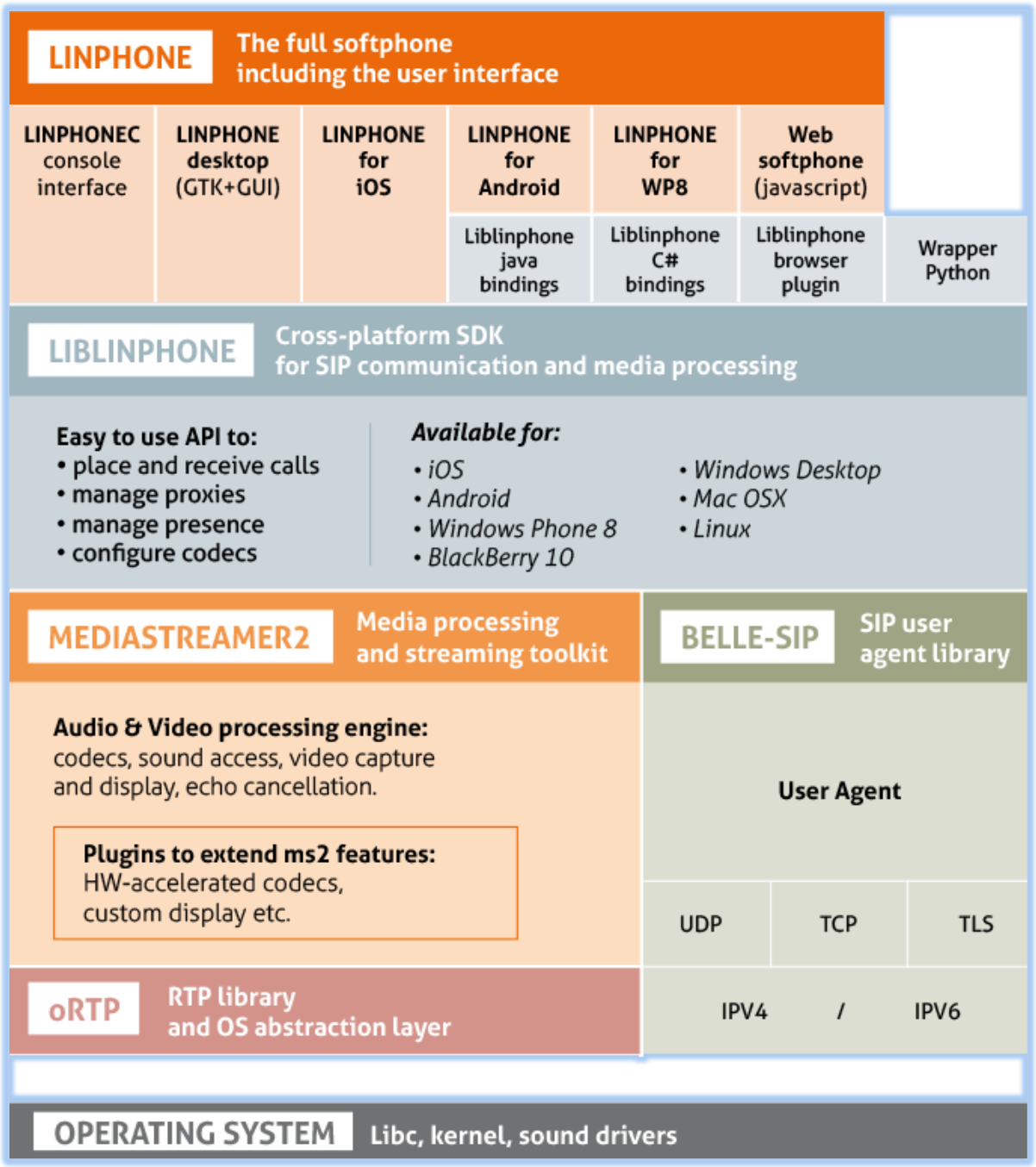
2. LINPHONE BASICS

2.1 INDRODUCTION

Linphone is an open source SIP Phone, available on mobile and desktop environments (iOS, Android, Windows Phone 8, Linux, Windows Desktop, MAC OSX) and on web browsers.

Linphone has inside a separation between the user interfaces and the core engine, allowing creating various kinds of user interface on top of the same functionalities.

2.2 ARCHITECTURE



2.2.1. LIBLINPHONE

Liblinphone is a high level, native library integrating all the SIP video calls feature into a single easy to use API. Usually telecommunications is made of two things: media (transport of voice or video, encoding and decoding...), and signaling (routing calls, ringing, accepting a call etc...).
More details about the Liblinphone could be found at LIBLINPHONE FEATURES.

2.2.2. LIBLINPHONE JAVA BINDINGS

In order to accommodate Liblinphone to the Android system, Linphone SDK has implemented Java bindings (i.e. wrappers) for Android. App developers can write their own SIP-based phone apps atop the Java bindings without having to re-implement all the SIP features in Java language again. Not only can the Java bindings save developers tons of time, but it also achieved the reusability of the Liblinphone native library.

3. LINPHONE SDK – GETTING STARTED

In this section, you will learn, step by step, about how to use Linphone SDK to add SIP phone features, including incoming and outgoing calls, for your Android app.

3.1 INITIALIZATION

3.1.1. IMPORT REQUIRED CLASSES

```
import org.linphone.LinphoneManager;
import org.linphone.LinphonePreferences;
import org.linphone.LinphoneService;
import org.linphone.SettingsFragment;
import org.linphone.core.LinphoneCall;
import org.linphone.core.LinphoneCore;
import org.linphone.core.LinphoneCoreException;
import org.linphone.core.LinphoneCoreListenerBase;
import org.linphone.core.PayloadType;
import org.linphone.core.LinphoneCore.EcCalibratorStatus;
import org.linphone.core.LinphoneCore.MediaEncryption;
import org.linphone.core.VideoSize;
```

3.1.2. START THE LINPHONE SERVICE

It's a good place to put the following code snippet in the onCreate method of your main activity.

```
if (!LinphoneService.isReady())  {
    startService(new Intent(ACTION_MAIN).setClass(this, LinphoneService.class));
}
```

3.1.3. WAIT THE LINPHONE SERVICE TO GET READY

You may check the readiness of the Linphone service in a timer task like this:

```
new Timer().schedule(new TimerTask() {

    @Override
    synchronized public void run() {
        if (LinphoneService.isReady()) {
            cancel();
            // get instances of LinphoneManager, LinphoneCore, LinphoneService!
            // set the activity to launch once an incoming call arrives
            // set preferences
            // enable required video codecs
            // enable required audio codecs
        }
    }

}, 3000, 1000);
```

3.1.4. GET SOME INSTANCES FROM THE LINPHONE SDK

You can get instances of LinphoneManager, LinphoneCore and LinphoneService in the way as follows:

```
LinphoneManager lm = LinphoneManager.getInstance();
LinphoneCore lc = lm.getLcIfManagerNotDestroyedOrNull();
LinphoneService ls = LinphoneService.instance();
```

3.1.5. SET THE ACTIVITY TO HANDLE INCOMING CALLS

If needed, you can designate an activity to handle incoming calls in place of the default one.

```
ls.setActivityToLaunchOnIncomingReceived(IncomingCallActivity.class);
```

3.1.6. PREFERENCES

You should properly set preferences to accommodate your app to your application. Here is an example used in the iTE Android app.

```
lc.setUserAgent("LinphoneAndroid", lc.getVersion());
mPrefs = LinphonePreferences.instance();
mPrefs.setDefaultDisplayName("android.app");
mPrefs.setDefaultUsername("app");
mPrefs.setSipPort(5060);
lc.setAudioPort(-1);
lc.setVideoPort(-1);
mPrefs.setMediaEncryption(MediaEncryption.None /*NONE/SRTP/ZRTP*/);
mPrefs.setPreferredVideoSize("720p" /*720p/vga/cif/qvga/qcif*/);
lc.setPreferredVideoSize(new VideoSize(720, 1280));
mPrefs.setEchoCancellation(true);
mPrefs.setFrontCamAsDefault(true);
mPrefs.enableVideo(true);
mPrefs.setFrontCamAsDefault(true);
lc.enableVideo(false,  true);
mPrefs.setInitiateVideoCall(true);
mPrefs.setPushNotificationEnabled(true);
mPrefs.setAutomaticallyAcceptVideoRequests(false);
mPrefs.setWifiOnlyEnabled(false);
mPrefs.useRandomPort(false);
mPrefs.setDebugEnabled(false);
mPrefs.setBackgroundModeEnabled(false);
mPrefs.setAutoStart(false);
```

3.1.7. VIDEO CODECS

We only interest in VP8 and H264 codecs.

```
String codecName;
for (PayloadType pt : lc.getVideoCodecs()) {
    codecName = pt.toString();
    String mime = pt.getMime().toUpperCase();
    try {
        if (mime.equals("VP8")
            || mime.equals("H264")) {
            lc.enablePayloadType(pt, true);
        }
        else {
            lc.enablePayloadType(pt, false);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

3.1.8. AUDIO CODECS

PCMA, PCMU, G722 and G729 with sampling rate less than 8KHz are what we need.

```
for (PayloadType pt : lc.getAudioCodecs()) {
    codecName = pt.toString();
    String mime = pt.getMime().toUpperCase();
    try {
        if (pt.getRate() <= 8000 && (mime.equals("PCMU")
                || mime.equals("PCMA")
                || mime.equals("G722")
                || mime.equals("G729")
                || mime.equals("SPEEX"))) {
            lc.enablePayloadType(pt, true);
        }
        else {
            lc.enablePayloadType(pt, false);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

3.2 DEINITIALIZATION

It's super easy to deinitialize the Linphone SDK – just try to stop the Linphone service.

```
if (LinphoneService.isReady()) {
    stopService(new Intent(ACTION MAIN).setClass(this, LinphoneService.class));
}
```

3.3 OUTGOING CALL BASICS

An outgoing call is composed of 2 things – placing a call and registering a listener to handle any call state events.

3.3.1. PLACING A DIRECT CALL

```
AddressType address = new AddressText(getActivity(), null);
address.setText("sip:toto@192.168.191.199");
LinphoneManager.getInstance().newOutgoingCall(address);
```

3.3.2. LINPHONE LISTENER

Start the InCallActivity activity

```
Intent intent = new Intent(getActivity(), InCallActivity.class);
intent.putExtra("VideoEnabled", true);
startActivity(intent);
```

In the onResume method, register a listener to the Linphone SDK to receive and handle any call state events.

```
LinphoneCore lc = LinphoneManager.getLcIfManagerNotDestroyedOrNull();
if (lc != null) {
    lc.addListener(new LinphoneCoreListenerBase(){
            @Override
        public void callState(LinphoneCore lc, final LinphoneCall call, LinphoneCall.State state, String
message) {
            if (LinphoneManager.getLc().getCallsNb() == 0) {
                finish();
                return;
            }

            if (state == State.IncomingReceived) {
                startIncomingCallActivity();
                return;
            }

            if (state == State.Paused || state == State.PausedByRemote ||  state == State.Pausing) {
                video.setEnabled(false);
                if(isVideoEnabled(call)){
                    showAudioView();
                }
            }

            if (state == State.Resuming) {
                if(LinphonePreferences.instance().isVideoEnabled()){
                    status.refreshStatusItems(call, isVideoEnabled(call));
                    if(call.getCurrentParamsCopy().getVideoEnabled()){
                        showVideoView();
                    }
                }
            }

            if (state == State.StreamsRunning) {
                switchVideo(isVideoEnabled(call));

                    //Check media in progress
                    if(LinphonePreferences.instance().isVideoEnabled() &&
                            !call.mediaInProgress()){
                        video.setEnabled(true);
                    }

                LinphoneManager.getLc().enableSpeaker(isSpeakerEnabled);
```

```
                isMicMuted = LinphoneManager.getLc().isMicMuted();
                enableAndRefreshInCallActions();

                if (status != null) {
                    videoProgress.setVisibility(View.GONE);
                    status.refreshStatusItems(call, isVideoEnabled(call));
                }
            }

            refreshInCallActions();

            refreshCallList(getResources());

            if (state == State.CallUpdatedByRemote) {
                // If the correspondent proposes video while audio call
                boolean videoEnabled = LinphonePreferences.instance().isVideoEnabled();
                if (!videoEnabled) {
                    acceptCallUpdate(false);
                    return;
                }

                boolean remoteVideo = call.getRemoteParams().getVideoEnabled();
                boolean localVideo = call.getCurrentParamsCopy().getVideoEnabled();
                boolean autoAcceptCameraPolicy =
                    LinphonePreferences.instance().shouldAutomaticallyAcceptVideoRequests();
                if (remoteVideo && !localVideo && !autoAcceptCameraPolicy &&
                        !LinphoneManager.getLc().isInConference()) {
                    showAcceptCallUpdateDialog();

                    timer = new CountDownTimer(SECONDS_BEFORE_DENYING_CALL_UPDATE, 1000) {
                        public void onTick(long millisUntilFinished) { }
                        public void onFinish() {
                                if (callUpdateDialog != null)
                                    callUpdateDialog.dismiss();
                            acceptCallUpdate(false);
                        }
                        }.start();
                    }
                }

                transfer.setEnabled(LinphoneManager.getLc().getCurrentCall() != null);
        }

        @Override
        public void callEncryptionChanged(LinphoneCore lc, final LinphoneCall call, boolean encrypted,
String authenticationToken) {
            if (status != null) {
                status.refreshStatusItems(call, call.getCurrentParamsCopy().getVideoEnabled());
            }
        }
    });
```

In the onPause method, remove the listener. It should look a little bit like this:

```
@Override
protected void onPause() {
        LinphoneCore lc = LinphoneManager.getLcIfManagerNotDestroyedOrNull();
        if (lc != null) {
            lc.removeListener(mListener);
        }

        super.onPause();
```

```
}
```

3.4 INCOMING CALLS

In 3.1.5, we have mentioned that you can set the activity to launch to handle any incoming calls. In the Linphone SDK, an activity, named IncomingCallActivity, has been implemented to handle all the things you need. This is the default activity that will handle incoming calls for your app. Your app got the ability to handle incoming calls without having to code a line. Super handy, isn't it?

3.5 WATCH

Just do the same things mentioned in 3.3 except for changing the SIP user name to "***camera***". The doorbell device will enter the watch mode automatically.

```
AddressType address = new AddressText(getActivity(), null);
address.setText("sip:camera@192.168.191.199");
LinphoneManager.getInstance().newOutgoingCall(address);
```

4. ITE-PROPRIETARY PROTOCOLS

4.1 OPEN THE DOOR

The following code snippet demonstrates how to open the door whose IP is “192.168.191.199” from an Android mobile device associated with an indoor phone whose ro is "01-02-01-01-01-01".

```
    import org.apache.http.HttpResponse;
    import org.apache.http.client.HttpClient;
    import org.apache.http.client.methods.HttpGet;
    import org.apache.http.impl.client.DefaultHttpClient;

…

    int statusCode = -1;
    HttpResponse httpResponse;
    HttpClient httpClient = new DefaultHttpClient();
    HttpGet httpRequest = new HttpGet(
                            “http://192.168.191.199/open door?ro=01-02-01-01-01-01”);

    try {
        httpResponse = httpClient.execute(httpRequest);
        statusCode = httpResponse.getStatusLine().getStatusCode();
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    getActivity().runOnUiThread(new Thread() {
        public void run() {
            View view;
            TextView text;
            Toast toast;

            toast = Toast.makeText(getActivity(), (statusCode == 200) ? R.string.opendoor o :
R.string.opendoor_x, Toast.LENGTH_SHORT);
            view = toast.getView();
            text = (TextView) view.findViewById(android.R.id.message);
            text.setTextColor(getResources().getColor((statusCode == 200) ? R.color.white :
R.color.red0));
            toast.setGravity(Gravity.CENTER, 0, 0);
            toast.show();
        }
    });
```