VoIP SIP SDK

DEVELOPER MANUAL

28.12.2019

Skype: voip.sip.sdk
Email: sales@voipsipsdk.com
P.: + 1-773-800-2879
www.voipsipsdk.com

Table of Contents

QUICK START

Using SDK:

VoIP SIP SDK allows to add audio/video call features to existing application, develop soft-phone with custom business logic and design, make calls automatically and implement many other cases.

App life cycle and implementation details:

SDK stars SIP stack in own background service. This allows app to update registration and received incoming calls in background and even when all activities closed (removed from list of recent apps).

App communicates with SDK via high level 'AbtoPhone' singleton instance, which is member of AbtoApplication.
This architecture allow access 'phone' in any place of app using code:

```
abtoPhone = ((AbtoApplication) getApplication()).getAbtoPhone();
```

App lifecycle is following:

1. Request runtime permissions on start;

2. Configure and initialize 'phone' instance;
   On this stage app binds to SDKs service. When service is already running app starts very fast.

3. Add SIP accounts (if required)
   SDK stores list of added accounts in local database and updates registration in service;
   Each time when registration update was successful or failed SDK sends notification to app;

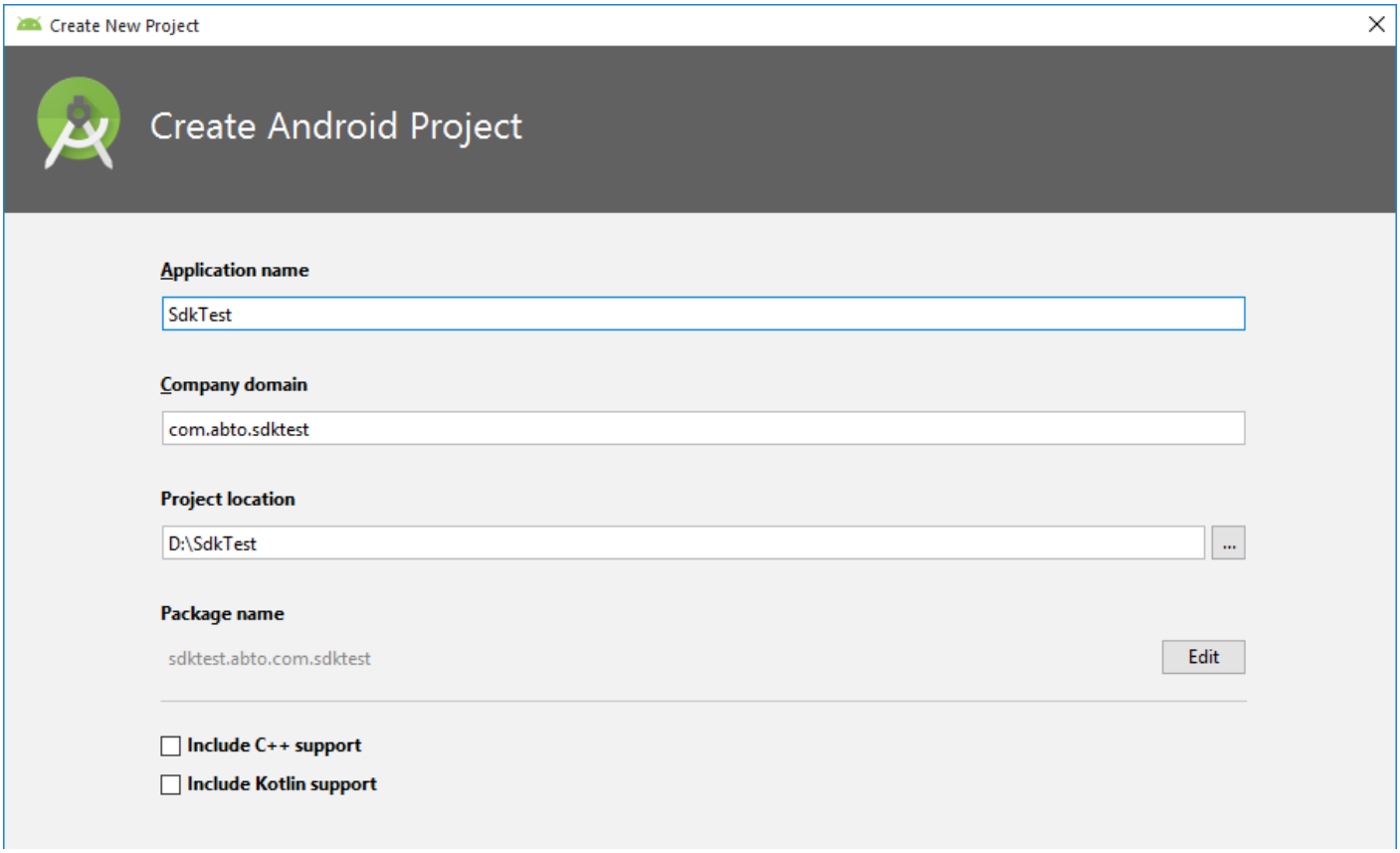
4. Make receive phone calls
   App can make outgoing/receive incoming call in any time after successful initialization and adding account;

5. Unregister and stop service

How to add SIP SDK to the application and make calls

Create new project

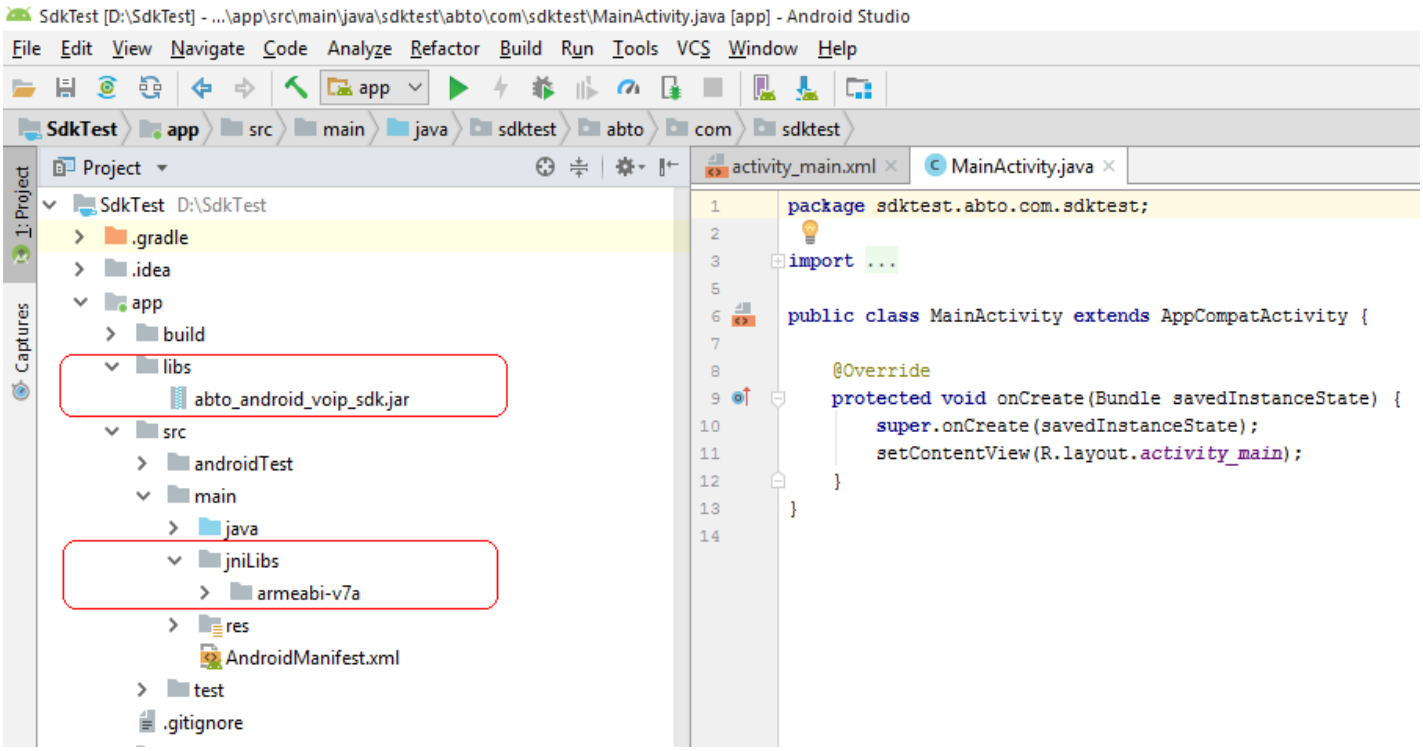
1. Open AndroidStudio IDE,
2. Click menu "File->New project"



3. Select "Phone and Tablet", select API version
4. Select "Empty Activity", continue next steps with default values

Copy SDK binary files

5. Copy SDK jar file to project:
   Go to folder with downloaded SDK and copy file
   `<SDK>\libs\abto_android_voip_sdk.jar`
   to:
   `<project>\app\libs\abto_android_voip_sdk.jar`
6. Copy SDK native libraries to project:
   Go to folder with downloaded SDK and copy folder
   `<SDK>\libs\armeabi-v7a`
   into:
   `<project>\app\src\main\jniLibs`



Edit manifest file:

a Modify application tag:

```
<application
      android:name="org.abtollc.sdk.AbtoApplication"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name">
```

b Add SDK service

```
<service
    android:name="org.abtollc.service.ABTOSipService"
    android:stopWithTask="true">
    <intent-filter>
        <action android:name="org.abtollc.service.ABTOSipService" />
        <action android:name="org.abtollc.service.SipConfiguration" />
    </intent-filter>
</service>
```

c Add db provider authorities (required for Android 7 and later)

```
<provider android:name="org.abtollc.db.DBProvider"
android:authorities="abto.com.sdktest"/>
<meta-data android:name="AbtoVoipAuthority" android:value="abto.com.sdktest" />
```

"abto.com.sdktest" means own package name.

Modify source code

7. Modify source code of existing Activity :

a Add 'import' directives:

```
import org.abtollc.sdk.AbtoApplication;
import org.abtollc.sdk.AbtoPhone;
import org.abtollc.sdk.AbtoPhoneCfg;
import org.abtollc.sdk.OnInitializeListener;
import org.abtollc.utils.Log;
import org.abtollc.utils.codec.Codec;
```

b Declare 'phone' member

```
AbtoPhone abtoPhone;
```

c Implement 'onCreate'

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Get AbtoPhone instance
    abtoPhone = ((AbtoApplication) getApplication()).getAbtoPhone();

    boolean bCanStartPhoneInitialization = (Build.VERSION.SDK_INT >= 23) ?
askPermissions() : true;

    if(bCanStartPhoneInitialization)    initPhone();
}
```

d Implement permissions request and SDK initialization

```
final private int REQUEST_CODE_ASK_MULTIPLE_PERMISSIONS = 124;

private boolean askPermissions()
{
    List<String> permissionsNeeded = new ArrayList<String>();

    final List<String> permissionsList = new ArrayList<String>();
    if (!addPermission(permissionsList, Manifest.permission.RECORD_AUDIO))
permissionsNeeded.add("Record audio");
    if (!addPermission(permissionsList,
Manifest.permission.WRITE_EXTERNAL_STORAGE)) permissionsNeeded.add("Write logs to
sd card");
    if (!addPermission(permissionsList, Manifest.permission.CAMERA))
permissionsNeeded.add("Camera");
    if (!addPermission(permissionsList, Manifest.permission.USE_SIP))
permissionsNeeded.add("Use SIP protocol");

    if (permissionsList.size() > 0) {
        if (permissionsNeeded.size() > 0) {
               ActivityCompat.requestPermissions(this,
                    permissionsList.toArray(new
String[permissionsList.size()]),
                    REQUEST_CODE_ASK_MULTIPLE_PERMISSIONS);

            return false;
        }

        ActivityCompat.requestPermissions(this,
                permissionsList.toArray(new String[permissionsList.size()]),
                REQUEST_CODE_ASK_MULTIPLE_PERMISSIONS);
        return false;
    }

    return true;
}

private boolean addPermission(List<String> permissionsList, String permission)
{
```

```
        if (ContextCompat.checkSelfPermission(this, permission) !=
PackageManager.PERMISSION_GRANTED) {
            permissionsList.add(permission);
            // Check for Rationale Option
            if (!ActivityCompat.shouldShowRequestPermissionRationale(this,
permission))     return false;
        }

        return true;
    }

    private void showMessageOKCancel(String message,
DialogInterface.OnClickListener okListener) {
        new AlertDialog.Builder(this)
                .setMessage(message).setPositiveButton("OK", okListener)
                .setNegativeButton("Cancel", null).create().show();
    }

    @TargetApi(23)
    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions,
int[] grantResults)
    {
        switch (requestCode)
        {
            case REQUEST_CODE_ASK_MULTIPLE_PERMISSIONS:
            {
                Map<String, Integer> perms = new HashMap<String, Integer>();
                //Initial
                perms.put(Manifest.permission.RECORD_AUDIO,
PackageManager.PERMISSION_GRANTED);
                perms.put(Manifest.permission.WRITE_EXTERNAL_STORAGE,
PackageManager.PERMISSION_GRANTED);
                perms.put(Manifest.permission.CAMERA,
PackageManager.PERMISSION_GRANTED);
                perms.put(Manifest.permission.USE_SIP,
PackageManager.PERMISSION_GRANTED);

                //Fill with results
                for (int i = 0; i < permissions.length; i++)
perms.put(permissions[i], grantResults[i]);

                //Check for ACCESS_FINE_LOCATION
                if (perms.get(Manifest.permission.RECORD_AUDIO) ==
PackageManager.PERMISSION_GRANTED
                        && perms.get(Manifest.permission.WRITE_EXTERNAL_STORAGE)
== PackageManager.PERMISSION_GRANTED
                        && perms.get(Manifest.permission.CAMERA) ==
PackageManager.PERMISSION_GRANTED
                        && perms.get(Manifest.permission.USE_SIP) ==
PackageManager.PERMISSION_GRANTED) {
                    // All Permissions Granted
                    initPhone();
                } else {
                    // Permission Denied
                    Toast.makeText(this, "Some permissions were denied",
Toast.LENGTH_SHORT).show();
                }
            }
            break;
            default:
                super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
        }
    }
```

e Implement 'phone' instance initialization

```
protected void initPhone()
{
    abtoPhone.setInitializeListener(this);

    AbtoPhoneCfg config = abtoPhone.getConfig();
    config.setCodecPriority(Codec.PCMU, (short) 200);
    config.setCodecPriority(Codec.PCMA, (short) 100);

    config.setCodecPriority(Codec.H264, (short) 220);
    config.setCodecPriority(Codec.H263_1998, (short) 210);

    config.setSipPort(0);
    config.setSignallingTransport(AbtoPhoneCfg.SignalingTransportType.UDP);

    Log.setLogLevel(5);
    Log.setUseFile(true);

    abtoPhone.initialize(true);
}
```

f Implement 'phone' instance initialization

AbtoPhone methods

Name	Description

initialize(boolean sticky)	This method initializes 'phone' instance.
	App has invoke it only once, when app started.
	
	Argument 'sticky' allows to control internal service, inside SDK.
	When app set this value to 'true', internal service will return 'START_STICKY' from 'onStartCommand'.
	`@Override`
	`public int onStartCommand(Intent intent, int flags, int startId) {`
	`  return START_STICKY;`
	`}`
	
	Which will cause that system will try to re-create service after its process was killed.
	
	During initialization SDK raises notification:
	`OnInitializeListener::onInitializeState(InitializeState state, String message)`
	
	When app received:
	`onInitializeState(SUCCESS, "Initialization success")`
	➔ It means, that SDK successfully initialized and
	
	When this notification doesn't received:
	➔ It means, that SDK can't be started because of missed native libraries for specific platforms or some other reason.
	Open Logcat window and review list of messages or copy and forward it to support team.
	➔
	Note:
	Service restarting depends on the manifest declaration:
	`<service android:name="org.abtollc.service.ABTOSipService" android:stopWithTask="true">`
	When service declared, as shown above, system will not restart it.

initialize()	This method does same as: `initialize(false)`
initializeForeground(Notification n)	This method added specifically for handling incoming calls via push notifications. It does same `initialize(false)`, but additionally displays notification, created by app and starts SDKs service in foreground mode using method 'startForegroundService'. When app received push notification it has initialize SDK and restore registration. On Android O and newer method 'initialize' causes `IllegalStateException "Not allowed to start service ABTOSipService app app is in background".`

	To fix this - app has invoke phone. initializeForeground(n).
	
	One more side effect of using this method – foreground service has higher priority and system will not stop it.
	
	Usage example:
	`//CreateChanel`
	`if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {`
	`NotificationChannel channel = new NotificationChannel(CHANEL_ID, getString(R.string.app_name), NotificationManager.IMPORTANCE_NONE);`
	`NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);`
	`if (notificationManager != null) notificationManager.createNotificationChannel(channel);`
	`}`
	
	`//Create intent`
	`Intent intent = new Intent(this, MainActivity.class);`
	`intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);`
	`intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);`
	`PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_CANCEL_CURRENT);`
	
	`//Create notification`
	`NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, CHANEL_ID);`
	`mBuilder.setAutoCancel(false);`
	`mBuilder.setOngoing(true);`
	`mBuilder.setOnlyAlertOnce(true);`
	`mBuilder.setContentIntent(pendingIntent);`
	`mBuilder.setContentText("ABTO SDK");`
	`mBuilder.setSubText("ABTO SDK");`
	`mBuilder.setSmallIcon(R.drawable.icon);`
	`Notification notification = mBuilder.build();`
	
	`//Initialize`
	`abtoPhone.initializeForeground(notification);`
	
	Note:
	Using foreground services requires to add permission in manifest:
	`<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />`

stopForeground()	This method hides notification, displayed by 'initializeForeground', but SDKs service continues working. App can invoke this method when Activity started or when user decided to end app.
register()	This method registers all accounts, added by method '`config.addAccount`'.

	App has invoke this method only after adding new account, in all other cases SDKs service updates registration automatically.
	
	App can detect account's registration status via handling notifications:
	`OnRegistrationListener::onRegistered(long accId)`
	`OnRegistrationListener::onRegistrationFailed(long accId, int statusCode, String statusText)`

unregister()	Method unregisters all previously added accounts. App can detect registration status via handling notification: `OnRegistrationListener::onUnRegistered(long accId)`
unregister(long accId)	Does the same as '**unregister**', but only for specified account. Argument 'accId' – account id, returned by 'config.addAccount'.

void startCall(String destination, long accountId)
 	
void startVideoCall(String destination, long accountId)
	Method starts (initiates) outgoing audio/video call.
	
	Arguments:
	'destination' – phone number or sip uri;
	'accountId' – account id, which account SDK has to use for this call.
	app can get this is by invoking 'phone.getActiveCallId()'
	
	When call established SDK sends notification:
	`OnCallConnectedListener::onCallConnected`
	
	When received "Trying"/"Ringing" messages, SDK raises notification:
	`OnRemoteAlertingListener::onRemoteAlerting`
	
	Example:
	`phone.startVideoCall("111", phone.getCurrentAccountId());`

hangUp(int status)	Method sends SIP BYE/CANCEL request, which ends (cancels) started call.
	
	Arguments:
	'status' – status code, which SDK will send to remote side.
	
	Call is successfully ended when received notification:
	`OnCallDisconnectedListener::onCallDisconnected`
	
	App can reduce time, how long SDK is waiting answer from remote side, using setting on initialization stage:
	`phone.getConfig()setHangupTimeout(400);//when answer on SIP BYE request didn't received during 400ms SDK will raise 'onCallDisconnected'.`

hangUp()	Does the same as above, sends default status code '486'.

answerCall(int status, boolean withVideo)	Method answers incoming call or enables early media.
	
	Arguments:
	'status' – status code, which SDK will send to remote side.
	'withVideo' - if 'true' SDK will add "video" media to SDP.
	
	Example:
	`phone.answerCall(200, true);//answer incoming call and send to remote side "200 OK" with 'video' media.`
	
	`phone.answerCall(183, false);//start early media and send to remote side "183 Session progress" without video.`

rejectCall()	Method rejects incoming call. Call is successfully ended when received notification: OnCallDisconnectedListener::**onCallDisconnected**
holdRetriveCall()	Method holds/unholds current call. Hold state means, that SDK stops send/receive RTP streams. When local or remote side put call on hold SDK raises notification: `OnCallHeldListener::onCallHeld`

playFile(String filePath, AbtoPhone.PlayFileWay way)	Method starts playing wav file to remote side during a call.
	Sound from file is mixed with sound from microphone.
	
	Arguments:
	'filePath' – path to wav file in device. SDK can't play file from 'asset' folder;
	'way' – this argument allows select where to play file (BOTH – play on local and remote side, REMOTE – play only to remote side);
	
	When playing finished SDK raises notification:
	`OnPlayFinishedListener::onPlayFinished`
	
	Example:
	`phone.playFile("/mnt/sdcard/AbtoVoIPClient/mozart.wav", AbtoPhone.PlayFileWay.BOTH);`

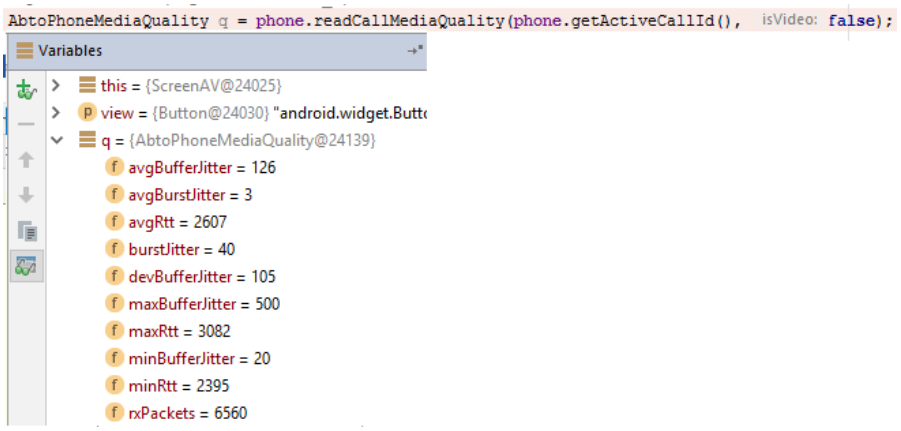
stopPlayback()	Stop playing file, started by '**playFile**'.
startRecording(String filePath)	Method starts recording sound of call (includes local and remote sound). Arguments: 'filePath' – path to wav file on device.
stopRecording()	Method stops recording, initiated by '**startRecording**'.

sendTone(char tone)
sendTone(int keyCode)
	Method sends DTMF tone to remote side.
	
	Arguments:
	'tone' – tone to send.
	Allowed values are: `'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '#', 'a', 'A', 'b', 'B', 'c', 'C', 'd', 'D', 'f', 'F'.`
	
	'keyCode' – tone to send.
	Allowed values are: `KeyEvent.KEYCODE_0.. KeyEvent.KEYCODE_9, KeyEvent.KEYCODE_STAR, KeyEvent.KEYCODE_POUND, KeyEvent.KEYCODE_A, KeyEvent.KEYCODE_B, KeyEvent.KEYCODE_C, KeyEvent.KEYCODE_D, KeyEvent.KEYCODE_F`
	
	Notes:
	- By default SDK tries to send tone as RTP signaling (RFC4733). When one of sides doesn't support this way SDK send tones as sound.
	- SDK allows to send sequence of tones via invoking this method few times.
	Example:
	`phone.sendTone('1');`
	`phone.sendTone('2');`
	`phone.sendTone('3');`

callXfer(String callee)	Method implements blind calls transfer.
	
	SDK sends SIP REFER request and reports status via notification:
	`OnCallTransferListener::onCallTransferState`
	
	Arguments:
	'callee' - fully qualified URI of remote side, where to transfer.
	
	Example:
	`phone.callXfer(sip:user@domain)`

sendTextMessage(long accountId, String msg, String destination)	Method sends text via SIP MESSAGE request.
	
	Arguments:
	'accountId' – account id, which account SDK has to use for this call. app can get use 'phone.getActiveCallId()'
	'msg' – text, which is required to send.
	'destination' – phone number or sip uri
	
	When call established SDK sends notification:
	`OnTextMessageStatusListener::onTextMessageStatus`
	
	Example:
	`phone.sendTextMessage(phone.getCurrentAccountId(), "hello", sipNumber);`

sendRequestNotify(long accountId, String body, String destination')	Method sends SIP NOTIFY request.
	
	Arguments:
	'accountId' – account id, which account SDK has to use for this call. app can get use 'phone.getActiveCallId()'
	'body' – text, which is required to copy as requests body.
	'destination' – phone number or sip uri
	
	When call established SDK sends notification:
	`OnTextMessageStatusListener::onTextMessageStatus`
	
	Example:
	`phone.sendTextMessage(phone.getCurrentAccountId(), "hello", sipNumber);`

readCallMediaQuality(int callId, boolean isVideo)	Method extracts RTP stream statistic of current call. 

setSpeakerLevel(float level)
setMicrophoneLevel(float level)
	Method modifies sound level using internal implementation, not related with system sound level.
	
	Argument:
	'level' – level to set.
	Allowed values:
	`0.0 - mutes speaker/microphone;`
	`0.1 - 0.9 - reduces sound volume;`
	`1.0 - set original volume;`
	`1.1 .. 30 - increase volume;`
	
	Example:
	`phone.setSpeakerLevel(2.0);`
	Increase speaker volume in 2 times

setMicrophoneMute(boolean on)	Method mutes local microphone
muteLocalVideo(boolean mute)	Method mutes local video (stops sending it to remote side);

setSpeakerphoneOn(boolean on)	Method routes sound of call to loud speaker;
setBluetoothOn(boolean on)	Method routes sound of call to paired Bluetooth device;
switchCameraToFront(boolean onFront)	Method switches front/back camera during established video call.
getActiveCallId()	Method returns callId of current call (started/initiated/incoming) or '`AbtoPhone.INVALID_CALL_ID`' when there is no call.
isVideoCall(int callId)	Method returns 'true' if current call has "video" media in SDP body of SIP request.
setVideoWindows(SurfaceView capturer, SurfaceView renderer)	Method allows to set 'SurfaceView' control, where SDK will draw local/remote video.
version()	Method returns string with version of currently used SDK.
getConfig()	Method returns "config" interface, which allows to add accounts and modify 'phone' settings.
destroy	Method stops SDKs service and destroys 'phone' instance.

AbtoPhoneCfg methods

Name	Description
addAccount	This method adds new SIP account and removes all previously added. Arguments: `String domain` – domain or IP address of SIP server; `String proxy` – address of remote, where to send SIP request. Typically is required when "domain" name of server can't be resolved via DNS. `String user` – user name (extension) which is required to register; `String pass` – registration password; `String authId` – is required when server requires specific user name for authentication, which differs from 'user'; `String displName` – display name (caller id) which SDK will add to SIP headers; `int expire` – time in seconds, how often SDK has to update account registration. Recommended values are in range [100 … 300]; `boolean enableDoubleRegistration` – this option allows enable sending registration requests twice. First time SDK sends REGISTER request with local IP:port in 'Contact' header, then resolves its public IP:port from servers response and sends registration second time with updated 'Contact' header. Return value: Method returns unique `accountId` value, which is required to use as argument in methods like `startCall(x, accountID)`. Also SDK uses accountId in notifications like `onRegistered(long accId)`.
addAccount	This method does exactly same as previous one, but has one more argument: `boolean unique` – when this argument is 'true' methods removes all existing accounts and adds this new one. When this argument is 'false' method just adds new account, which allows to have few SIP accounts.

setContactDetails
setContactDetailsUri
	These methods allow to set specific strings, which SDK will add into 'Contact' header of SIP REGISTER request. Typically it's required for send push token to server.
	
	Usage example:
	`AbtoPhoneCfg config = abtoPhone.getConfig();`
	`config.setContactDetailsUri(";detailsUri");`
	`config.setContactDetails(";token="+URLEncoder.encode("yyyy+xxx 111 <1>"));`
	`long accId = config.addAccount(RegDomain, null, RegUser, RegPassword, null, "", regTimeout, false);`
	
	Code above produces following 'Contact' header:
	`Contact: <sip:user@domain;detailsUri>;token=yyyy+xxx+111+%3c1%3e`

	Note: Using wrong symbols in input string may block sending registration request. In case of issues with sending registration make these values empty and send registration again.
setSTUNEnabled isSTUNEnabled	This method enables/disable to use STUN server, configure by 'setSTUNServer'.

getSTUNServer
 	
setSTUNServer
	This method set STUN server, which SDK will use to resolve public IP:port
	Example:
	`config.setSTUNEnabled(true);`
	`config.setSTUNServer("stun1.l.google.com:19302");`

setSignallingTransport	This method allows set signaling transport which SDK has to use for send/receive SIP requests.
	Available transports:
	`AbtoPhoneCfg.SignalingTransportType.UDP`
	`AbtoPhoneCfg.SignalingTransportType.TCP`
	`AbtoPhoneCfg.SignalingTransportType.TLS`
	
	Example:
	`config.setSignallingTransport(AbtoPhoneCfg.SignalingTransportType.TLS);`

setSipPort	This method allows set **local** port number, which SDK bind on device.
	
	Example:
	`config.setSipPort(0);`
	Value 0 – means using random port number.
	
	Note 1:
	We don't suggest to use local port number 5060.
	On some devices it's blocked by system (is allows to bind to this port, but doesn't allowed to send receive data)
	One more possible issue with using this port number – app which is working on device with public IP address may receive unwanted incoming calls or event registration request from Internet.
	
	Note 2
	When is required to send request to specific port number on server side – set this pot as part of domain or proxy name.
	Example – app has sends SIP request to port 5088 on some server.
	Configure this using code:
	`String domain = "x.x.x.x:5088";`
	`config.addAccount(domain, …)`

setTLSVerifyServer	This method disables/enables verification certificate, received from server on establishing TLS connection.
	
	Example 1:
	`config.setSignallingTransport(SignalingTransportType.TLS);`
	`config.setTLSVerifyServer(false);`

	In this mode SDK will always accept TLS connection with remote server. This mode recommended when app directly uses IP address of SIP server without its domain name.

setTLSCAListFile	This method allows set CA certificate which SDK will use to verify servers certificate before establish TLS connection. In this mode connection will be established only when servers certificate signed by CA cert AND 'CN' value in received servers cert is same as 'Domain' of registering account.
	
	Note:
	SDK requires to have path to cert file, copied on device.
	
	Example
	`AbtoPhoneCfg config = abtoPhone.getConfig();`
	`...`
	`String path = Environment.getExternalStorageDirectory() + "/." + appName;`
	`config.setTLSCAListFile(pathPrefix + "/cacert.pem");`
	`config.setTLSVerifyServer(true);`
	`...`
	`abtoPhone.initialize(true);`

setEnableSipsSchemeUse	This method allows to use '`sip`' scheme instead of default '`sips`' when TLS transport enabled.
setICEEnabled	This method allows enable ICE technique. Should be also enabled STUN. https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment

setCodecPriority	This method allows configure priority of codecs which SDK will send in SDP body of SIP INVITE/OK requests.
	
	Priority values are related and means, that codec with higher value will be added on top of list.
	0 – mean remove codec from SDP.
	
	Example:
	`AbtoPhoneCfg config = abtoPhone.getConfig();`
	`config.setCodecPriority(Codec.G729, (short) 0);`
	`config.setCodecPriority(Codec.GSM, (short) 0);`
	`config.setCodecPriority(Codec.PCMU, (short) 200);`
	`config.setCodecPriority(Codec.PCMA, (short) 100);`
	
	`config.setCodecPriority(Codec.H264, (short) 220);`
	`config.setCodecPriority(Codec.H263_1998, (short) 210);`
	
	Code above disables "G729" and "GSM" and enables PCMU, PCMA, H264, H263_1998 and invoking phone.startVideoCall() will generated following SDP:
	
	`m=audio 12002 RTP/AVP 0 8 101`
	`c=IN IP4 172.30.20.222`
	`a=rtcp:12003 IN IP4 172.30.20.222`
	`a=sendrecv`
	`a=rtpmap:0 PCMU/8000`
	`a=rtpmap:8 PCMA/8000`

	a=rtpmap:101 telephone-event/8000 a=fmtp:101 0-15 m=video 12004 RTP/AVP 97 96 34 c=IN IP4 172.30.20.222 a=rtcp:12005 IN IP4 172.30.20.222 a=sendrecv a=rtpmap:97 H264/90000 a=fmtp:97 profile-level-id=42C014 a=rtpmap:96 H263-1998/90000 a=fmtp:96 CIF=1;QCIF=1 a=rtpmap:34 H263/90000 a=fmtp:34 CIF=1;QCIF=1 Note: We suggest to explicitly set priorities of all codecs.
setUserAgent	Method allows set string, which SDK will copy to SIP header 'User-Agent' Example: User-Agent: ABTOLLC
setRtpPort	Method allows set start port of range of UDP ports which SDK uses for sending audio/video RTP/RTCP packets.
enableRingtone	Method enables/disables playing ringtone when incoming call received. When this option enabled – SDK will also star vibration, when it's selected in device setting. When this option disabled – SDK will not start vibration.
setRingtone	Method set ringtone wav file, which SDK has to play when incoming call received. SDK requires to use file on device and can't play files from "asset" folder. Example: config.enableRingtone(true); String ringtonefile = "file://" + getExternalStorageDirectory().toString() +"/com.abtotest.voiptest/mozart.wav"; config.**setRingtone**(ringtonefile);
enableRingbacktone	Method enables playing "long beep" tones when SDK makes outgoing call.
setUseSRTP	Method enables audio/video encryption using SRTP protocol.
setUseZRTP	Method enables audio/video encryption using ZRTP protocol.
setKeepAliveInterval	Method enables sending short keep-alive packets, which prevent closing UDP ports on routers between app and SIP server because of inactivity. Example: config.setSignallingTransport(AbtoPhoneCfg.SignalingTransportType.UDP) config.setKeepAliveInterval(AbtoPhoneCfg.SignalingTransportType.UDP, 30); Send keep-alive each 30seconds.

setRegisterTimeout	Method allows set delay in ms, how long SDK will wait answer from server on sent SIP REGISTER request.
	
	Example:
	`config.setRegisterTimeout(3000);`
	When answer on REGISTER did't received during 3000ms SDK will raise
	`onRegistrationFailed(accId, 408, ..)`
	
	Note:
	"RegisterTimeout" and "RegExpire" value are completely different.

setHangupTimeout	Method allows set delay in ms, how long SDK will wait answer from server on sent SIP BYE or SIP CANCEL request.

setVideoQualityMode	Method allows set video resolution, which SDK will use when capture image from local camera.
	Available modes:
	`enum AbtoPhoneCfg.VIDEO_QUALITY_MODE`
	`{`
	`VIDEO_MODE_DEFAULT,`
	`VIDEO_MODE_352_288,`
	`VIDEO_MODE_720_480,`
	`VIDEO_MODE_1280_720,`
	`VIDEO_MODE_1920_1080,`
	`VIDEO_MODE_352_288_PORTRAIT,`
	`VIDEO_MODE_720_480_PORTRAIT,`
	`VIDEO_MODE_176_144,`
	`VIDEO_MODE_176_144_PORTRAIT;`
	`}`
	
	Default value is: `VIDEO_MODE_352_288;`

setVideoFps	Method allows local capturer frame rate (how many frames per second has SDK send to remote side). Default value is - 15;

setEC	Method modifies echo cancelation settings.
	`setEC(int mode, int tailLengthMs)`
	
	Arguments:
	Mode - should be one of: Auto(0), Simple(1), Speex(2), Webrtc(3), default value is 3
	tailLengthMs - should be in range 50..300, default value = 200.
	
	Example:
	Good results reported with following configuration:
	`config.setEC(0, 0);`
	`config.setEC(3, 200);`

setEnableAutoSendRtpVideo	Method allows disable sending video RTP automatically when call started.
	Usage example:
	App starts video call and is able to see remote video, but don't want to send own video.
	
	To do this set option:
	`config.setEnableAutoSendRtpVideo(false);`
	
	To start sending video manually invoke:
	`abtoPhone.muteLocalVideo(false);`

setEnableAutoSendRtpAudio	Method does the same as "setEnableAutoSendRtpVideo" with audio RTP.
setMwiEnabled()	Methods enables sending SIP SUBSCRIBE requests and retrieve MWI data from server side.

setLogLevel	Method configures log details.
	`setLogLevel(int level, boolean toFile)`
	level – int value in range [0..5]
	toFile – enable/disable write logs to files on device.
	
	Example:
	`config.setLogLevel(5, true);`
	
	Set most detailed log level and enables write logs to files, created on device in folder "`<package_name>\logs`". Each time when app starts it creates new log file.

AbtoPhone callbacks

Name	Description

	SDK has set of callbacks/events and sends them in different cases.
	App has subscribe notification receiving via invoking method like:
	`phone.set<>Listener(classWhichImplementsInterface);`
	
	Note:
	SDK allows to set only one listener and doesn't keep set of listeners.
	
	Example 1:
	App invokes
	`phone.setInitializeListener(activity1);`
	`phone.setInitializeListener(activity2);`
	Now only 'activity2' can receive 'onInitializeState' notification.
	
	Example 2:
	`phone.setInitializeListener(activity1);`
	`phone.setInitializeListener(null);`
	Now no one can receive 'onInitializeState' notification.

onInitializeState(InitializeState state, String message)	SDK sends this notifications on initialization stage, which happens when app started first time or device lost and restored network connection.
	
	To receive this notification app has invoke:
	`phone.setInitializeListener(class which implement OnInitializeListener);`
	
	Possible 'state' values are:
	`enum InitializeState{`
	`START(1), INFO(2), WARNING(3), SUCCESS(4), FAIL(5)`
	
	Value 'SUCCESS' means, that SIP stack is initialized and app can add accounts or start call.

void onRegistered(long accId)	SDK sends this notifications when SIP account successfully registered. To receive this notification app has invoke: `phone.setRegistrationStateListener(this);`
void onUnRegistered(long accId)	SDK sends this notifications when SIP account successfully un-registered. To receive this notification app has invoke: `phone.setRegistrationStateListener(this);`
void onRegistrationFailed(long accId, int statusCode, String statusText)	SDK sends this notifications when registration failed. Arguments 'statusCode' and 'statusText' allows detect error code, received from remote side and reason of issue. To receive this notification app has invoke: `phone.setRegistrationStateListener(this);`

onNetworkStateChanged(boolean connected, String networkType)	SDK sends this notifications when lost/restored network connection. Argument 'connected' – is true when connection restored; Argument 'networkType' – contains name of network, like: "WiFi", "Edge", "3G". To receive this notification app has invoke: `abtoPhone.setNetworkEventListener();`
onRemoteAlerting(long accId, int statusCode)	SDK sends this notifications when outgoing call started and from remote side received SIP message "100Trying", "180 Ringing", "183 Session progress". Argument 'statusCode' – contains status code, received from remote side. Like: 100, 180, 183. To receive this notification app has invoke: `abtoPhone.setRemoteAlertingListener();`
onCallConnected(String remoteContact)	SDK sends this notification when outgoing call answered on remote side or incoming call answered locally. To receive this notification app has invoke: `abtoPhone.setCallConnectedListener();`
onCallDisconnected(String remoteContact, int callId, int statusCode)	SDK sends this notification when: - outgoing call rejected on remote side - incoming call rejected locally. - successfully established call ended on local or remote side argument '`remoteContact`' – contains data about remote contact argument '`callId`' – unique id of this call argument '`statusCode`' – status code, receive from remote side. To receive this notification app has invoke: `abtoPhone.setCallDisconnectedListener(OnCallDisconnectedListener)`
onCallDisconnected(String remoteContact, int callId, int statusCode, String sip)	This notification is same as above, but has one more argument: String sip – whole SIP request, received from remote side. To receive this notification app has invoke: `abtoPhone.setCallDisconnectedListener(OnCallDisconnectedListener2)`
onCallError(String remoteContact, int statusCode, String message)	SDK sends this notification when outgoing call failed. Arguments 'statusCode' and 'message' contains data about error. To receive this notification app has invoke: `abtoPhone.setCallErrorListener();`
onCallHeld(HoldState state)	SDK sends this notification when local or remote side put call on hold. Argument 'state' allows detect current state of established call. It can be one of: ACTIVE(1), LOCAL_HOLD(2), REMOTE_HOLD(3), ERROR(4); To receive this notification app has invoke: `abtoPhone.setOnCallHeldListener();`

onReceivedSipNotifyMsg(String msg)	SDK sends this notification when received SIP NOTIFY request from remote side. Argument msg – contains received request as string. To receive this notification app has invoke: `abtoPhone.setNotifyEventListener();`
onPlayFinished()	SDK sends this notification when finished playing file, which was started by method `phone. playFile();` To receive this notification app has invoke: `abtoPhone.setPlayFinishedListener();`
onMwiInfo(long accId, String mime, String text)	SDK sends this notification when received SIP NOTIFY request with MWI data. Argument "text" contains body of received request. To receive this notification app has invoke: `abtoPhone.setMwiEventListener();`
onToneReceived(char tone)	SDK sends this notification when received DTMF tone from remote side. Argument 'tone' – contains received tone. To receive this notification app has invoke: `abtoPhone.setToneReceivedListener();`
onTextMessageReceived(String message, String remoteContact, String accId)	SDK sends this notification when received SIP MESSAGE request from remote side. To receive this notification app has invoke: `abtoPhone.setInMessageListener();`
onTextMessageStatus(SipMessage msg)	SDK sends this notification when received response on SIP MESSAGE request, sent by method `phone.SendTextMessage` To receive this notification app has invoke: `abtoPhone.setTextMessageStatusListener()`
onRenderResolutionChanged(int callId, int width, int height)	SDK sends this notification when received and successfully decode first video frame from remote side. This notification allows properly resize and align surface view control, where SDK renders received video. To receive this notification app has invoke: `abtoPhone.setVideoEventListener();`
onCallTransferRequest(int callId, String remoteContact)	SDK sends this notification when received SIP REFER request from remote side. To receive this notification app has invoke: `abtoPhone.setCallTransferListener();`
onCallTransferState(int callId, int statusCode, String statusText)	SDK sends this notification when received status responses from remote side on SIP REFER request send by SDK using method '`phone.callXfer()`'. Arguments 'statusCode' and 'statusText' allows detect call transfer status. To receive this notification app has invoke: `abtoPhone.setCallTransferListener();`

OnIncomingCall(String remoteContact, long accountId)	SDK sends this notification when received incoming call. Argument 'remoteContact' contains data about caller. In case when app removed from list of recent apps it will be not able to receive this notification and answer or reject incoming call. Check detailed explanation how to do this in section "How to receive incoming calls" below. To receive this notification app has invoke: `abtoPhone.setIncomingCallListener(OnIncomingCallListener);`
OnIncomingCall(String remoteContact, long accountId, String sip)	This notification is same as above, but has additional argument 'sip', which contains SIP request, received from remote side. To receive this notification app has invoke: `abtoPhone.setIncomingCallListener(OnIncomingCallListener2);`

FAQ. PROBLEMS AND SOLUTIONS

How configure SDK service to restart automatically

Declare SDKs service in manifest file with flag '**stopWithTask = false**

```
<service
  android:name="org.abtollc.service.ABTOSipService"
  android:stopWithTask="false">
```

Initialize SDK using method: `phone.initialize(true);`

When app has to make outgoing calls only and running in background is not required declare service with flag "**stopWithTask = true**'

How to prevent system from stopping SDK service in background

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    Intent intent = new Intent();
    intent.setAction(Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS);
    intent.setData(Uri.parse("package:" + getPackageName()));
    startActivity(intent);
}
```

How to receive incoming calls

When service is running in background and incoming call received it notifies app about this event via sending broadcast message. App has receive that message and displays notification (Android doesn't allow to display Activity from background service).
We suggest to use following implementation (already added in example app):

Add CallEventsReceiver class

```
public class CallEventsReceiver extends BroadcastReceiver {

@Override
public void onReceive(Context context, Intent intent) {
Bundle bundle = intent.getExtras();
if ( bundle == null ) return;

if ( bundle.getBoolean(AbtoPhone.IS_INCOMING, false) ) {

// Incoming call
buildIncomingCallNotification(context, bundle);

} else if ( bundle.getBoolean(KEY_REJECT_CALL, false) ) {

// Reject call
int callId = bundle.getInt(AbtoPhone.CALL_ID);
CallEventsReceiver.cancelIncCallNotification(context, callId);
try {
App.getApp().getAbtoPhone().rejectCall();
} catch (RemoteException e) {
e.printStackTrace();
}

} else if ( bundle.getInt(AbtoPhone.CODE) == -1 ) {

// Cancel call
```

```
int callId = bundle.getInt(AbtoPhone.CALL_ID);
cancelIncCallNotification(context, callId);
}
}

public static final String CHANEL_CALL_ID = "abto_phone_call";
private static NotificationChannel channelCall;
public static final String KEY_PICK_UP_AUDIO = "KEY_PICK_UP_AUDIO";
public static final String KEY_PICK_UP_VIDEO = "KEY_PICK_UP_VIDEO";
public static final String KEY_REJECT_CALL = "KEY_REJECT_CALL";
private static final int NOTIFICATION_INCOMING_CALL_ID = 1000;

private void buildIncomingCallNotification(Context context, Bundle bundle) {
Intent intent = new Intent(context, ScreenAV.class);
intent.putExtras(bundle);

if ( !App.getApp().isAppInBackground() ) {//App is foreground - start activity
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startActivity(intent);
return;
}

boolean isVideoCall = bundle.getBoolean(AbtoPhone.HAS_VIDEO, false);
String title= isVideoCall ? "Incoming video call" : "Incoming call";
String remoteContact= bundle.getString(AbtoPhone.REMOTE_CONTACT);
int callId  = bundle.getInt(AbtoPhone.CALL_ID);

// Create channel
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O && channelCall == null) {
channelCall = new NotificationChannel(CHANEL_CALL_ID,
context.getString(R.string.app_name) + " Call", NotificationManager.IMPORTANCE_HIGH);
channelCall.setDescription(context.getString(R.string.app_name));
NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
if (notificationManager != null) {
notificationManager.createNotificationChannel(channelCall);
}
}

// Intent for launch ScreenAV
PendingIntent notificationPendingIntent = PendingIntent.getActivity(context, 1, intent,
PendingIntent.FLAG_UPDATE_CURRENT);

// Intent for pickup audio call
Intent pickUpAudioIntent = new Intent(context, ScreenAV.class);
pickUpAudioIntent.putExtras(bundle);
pickUpAudioIntent.putExtra(KEY_PICK_UP_AUDIO, true);
PendingIntent pickUpAudioPendingIntent = PendingIntent.getActivity(context, 2,
pickUpAudioIntent, PendingIntent.FLAG_UPDATE_CURRENT);

// Intent for reject call
Intent rejectCallIntent = new Intent();
rejectCallIntent.setPackage(context.getPackageName());
rejectCallIntent.setAction(AbtoPhone.ACTION_ABTO_CALL_EVENT);
rejectCallIntent.putExtra(AbtoPhone.CALL_ID, callId);
rejectCallIntent.putExtra(KEY_REJECT_CALL, true);
PendingIntent pendingRejectCall = PendingIntent.getBroadcast(context, 4,
rejectCallIntent, PendingIntent.FLAG_CANCEL_CURRENT);

// Style for popup notification
NotificationCompat.BigTextStyle bigText = new NotificationCompat.BigTextStyle();
bigText.bigText(remoteContact);
bigText.setBigContentTitle(title);
```

```
// Create notification
NotificationCompat.Builder builder = new NotificationCompat.Builder(context,
CHANEL_CALL_ID);
builder.setSmallIcon(R.drawable.ic_notif_pick_up_audio)
.setColor(0xff00ff00)
.setContentTitle(title)
.setContentIntent(notificationPendingIntent)
.setContentText(remoteContact)
.setDefaults(Notification.DEFAULT_ALL)
.setStyle(bigText)
.setPriority(Notification.PRIORITY_MAX)
.setCategory(NotificationCompat.CATEGORY_CALL)
.addAction(R.drawable.ic_notif_cancel_call, "Hang Up", pendingRejectCall)
.addAction(R.drawable.ic_notif_pick_up_audio, "Audio", pickUpAudioPendingIntent)
.setFullScreenIntent(notificationPendingIntent, true);

if (isVideoCall) {
// Intent for pickup video call
Intent pickUpVideoIntent = new Intent(context, ScreenAV.class);
pickUpVideoIntent.putExtras(bundle);
pickUpVideoIntent.putExtra(KEY_PICK_UP_VIDEO, true);
PendingIntent pickUpVideoPendingIntent = PendingIntent.getActivity(context, 3,
pickUpVideoIntent, PendingIntent.FLAG_UPDATE_CURRENT);
builder.addAction(R.drawable.ic_notif_pick_up_video, "Video", pickUpVideoPendingIntent);
}

NotificationManager mNotificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
Notification notification = builder.build();
mNotificationManager.notify(NOTIFICATION_INCOMING_CALL_ID + callId, notification);
}

public static void cancelIncCallNotification(Context context, int callId) {
NotificationManager mNotificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
if (mNotificationManager != null) {
mNotificationManager.cancel(NOTIFICATION_INCOMING_CALL_ID + callId);
}
}
}
```

Add USE_FULL_SCREEN_INTENT permission to manifest

```
<uses-permission android:name="android.permission.USE_FULL_SCREEN_INTENT" />
```

Add AppInBackgroundHandler class

```
public class AppInBackgroundHandler implements Application.ActivityLifecycleCallbacks {

private int activeActivities = 0;

@Override
public void onActivityCreated(@NonNull Activity activity, @Nullable Bundle bundle) { }

@Override
public void onActivityStarted(@NonNull Activity activity) {  }

@Override
public void onActivityResumed(@NonNull Activity activity) {
activeActivities++;
}

@Override
public void onActivityPaused(@NonNull Activity activity) {
```

```
activeActivities--;
}

@Override
public void onActivityStopped(@NonNull Activity activity) { }

@Override
public void onActivitySaveInstanceState(@NonNull Activity activity, @NonNull Bundle
bundle) { }

@Override
public void onActivityDestroyed(@NonNull Activity activity) { }

public boolean isAppInBackground() { return activeActivities == 0; }
}
```

Register receivers and App lifecycle handler in application class

```
public class App extends AbtoApplication {

private static App app;

private CallEventsReceiver callEventsReceiver = new CallEventsReceiver();
private AppInBackgroundHandler appInBackgroundHandler;

@Override
public void onCreate() {
super.onCreate();
app = this;
registerReceiver(callEventsReceiver, new IntentFilter(AbtoPhone.ACTION_ABTO_CALL_EVENT));

appInBackgroundHandler = new AppInBackgroundHandler();
registerActivityLifecycleCallbacks(appInBackgroundHandler);
}

@Override
public void onTerminate() {
super.onTerminate();
unregisterReceiver(callEventsReceiver);
unregisterActivityLifecycleCallbacks(appInBackgroundHandler);
}

public static App getApp() {
return app;
}

public boolean isAppInBackground() {
return appInBackgroundHandler.isAppInBackground();
}
}
```

Modify onCreate() method of CallInProgress activity:

Activity has verify – if was is started by service is required to initialize 'phone' instance (bind it to running service).

```
//Verify mode, in which was started this activity
boolean bIsIncoming= getIntent().getBooleanExtra(AbtoPhone.IS_INCOMING, false);
            boolean startedFromService =
getIntent().getBooleanExtra(AbtoPhone.ABTO_SERVICE_MARKER, false);
            if (startedFromService) {
                  phone.initialize(true);
                  phone.setInitializeListener(this);
```

```
            } else {
                  answerCallByIntent();
            }
```

How to handle device rotations and rotate video during a call

Set android:screenOrientation="portrait" for CallActivity in Manifest

```
<activity android:name=".CallActivity"
android:excludeFromRecents="true"
android:screenOrientation="portrait"
android:launchMode="singleTask" />
```

Add following code in CallActivity:

```
      private OrientationEventListener rotationListener;
      private Point videoViewSize;

      @Override
      public void onCreate(Bundle savedInstanceState) {

            ...


            // Init render resolution changed listener
            phone.setVideoEventListener(new OnVideoEventListener() {
                  @Override
                  public void onRenderResolutionChanged(int callId, int width, int
height) {
                        videoViewSize = new Point(width, height);
                        resizeRemoteVideoWindow(width, height, false);
                  }
            });

            // Init screen orientation event listener
            rotationListener = new OrientationEventListener(this) {
                  private AbtoPhone.Rotation angle = AbtoPhone.Rotation.ROTATION_0;

                  @Override
                  public void onOrientationChanged(int orientation) {
                        try {
                              AbtoPhone.Rotation rotation =
degreeToRotation((orientation + 45) / 90 * 90);
                              if ( angle != rotation ) {

                                    AbtoPhone phone = App.getApp().getAbtoPhone();
                                    phone.rotateCapturer(rotation);
                                    phone.rotateLocalVideo(rotation);
                                    phone.rotateRemoteVideo(rotation);

                                    angle = rotation;

                                    // resize video view
                                    boolean landscapeOrientation = angle ==
AbtoPhone.Rotation.ROTATION_90 ||
angle == AbtoPhone.Rotation.ROTATION_270;
                                    if ( videoViewSize != null ) {
                                          resizeRemoteVideoWindow(videoViewSize.x,
videoViewSize.y, landscapeOrientation);
                                    }
                              }
                        } catch (RemoteException e) {
```

```
                        e.printStackTrace();
                    }
                }

                private AbtoPhone.Rotation degreeToRotation(int orientation) {
                    if (orientation >= 60 && orientation <= 140) {
                        return AbtoPhone.Rotation.ROTATION_90;
                    } else if (orientation >= 140 && orientation <= 220) {
                        return AbtoPhone.Rotation.ROTATION_180;
                    } else if (orientation >= 220 && orientation <= 300) {
                        return AbtoPhone.Rotation.ROTATION_270;
                    }

                    return AbtoPhone.Rotation.ROTATION_0;
                }
            };

            // Enable listener
            rotationListener.enable();
        }

        public void onDestroy() {

            ...

            // Disable listener
            rotationListener.disable();
        }


        public void resizeRemoteVideoWindow(int videoW, int videoH, boolean
landscapeOrientation) {
            // Get screen size
            Point point = new Point();
            getWindowManager().getDefaultDisplay().getSize(point);
            int screenW = point.x;
            int screenH = point.y;

            if ( landscapeOrientation ) {
                // Switch remote video width with height for landscape orientation
                int temp = videoW;
                videoW = videoH;
                videoH = temp;
            }

            float videoSideKof = videoW / (float)videoH;
            float screenSideKof = screenW / (float)screenH;

            int viewW;
            int viewH;

            if ( videoSideKof > screenSideKof ) {
                viewW = screenW;
                viewH = (int) (viewW / (float)videoW * videoH);
            } else {
                viewH = screenH;
                viewW = (int) (viewH / (float)videoH * videoW);
            }

            FrameLayout.LayoutParams params = (FrameLayout.LayoutParams)
remoteVideoSurface.getLayoutParams();
            params.width = viewW;
            params.height = viewH;
            remoteVideoSurface.setLayoutParams(params);
```

```
}
```

How to update SDK settings in runtime

When is required to modify SDK settings like port number, codecs, etc in runtime modify them, using same methods as on initialization stage, and invoke:

```
try {
    abtoPhone.restartSip();
    Log.e(TAG, "SETTINGS SAVED");
} catch (NullPointerException e) {
    e.printStackTrace();
    Log.e(TAG, "SETTINGS ERROR");
}
```

How to register few SIP accounts

SDK allows to add and register few SIP accounts.
This can be done in following way:

```
AbtoPhoneCfg cfg = abtoPhone.getConfig();
accId1 = cfg.AddAccount(domain1, null, user1, pass1, null, "", 300, false, true);
accId2 = cfg.AddAccount(domain2, null, user2, pass2, null, "", 300, false, false);
accId3 = cfg.AddAccount(domain3, null, user3, pass3, null, "", 300, false, false);

//Register added accounts
try
{
   abtoPhone.Register();
}
catch (RemoteException ex)
{
   ex.PrintStackTrace();
}
```

When is required to make outgoing call from selected account use AccountId, returned by 'AddAccount'.

```
phone.startCall(sipNumber, accId3);
```

When received incoming call or registration events SDK sends 'AccountId' as argument, which allows to detect account of this event.

```
void onRegistrationFailed(long accId, int statusCode, String statusText)
```