How to add ABTO VoIP SDK to existing project

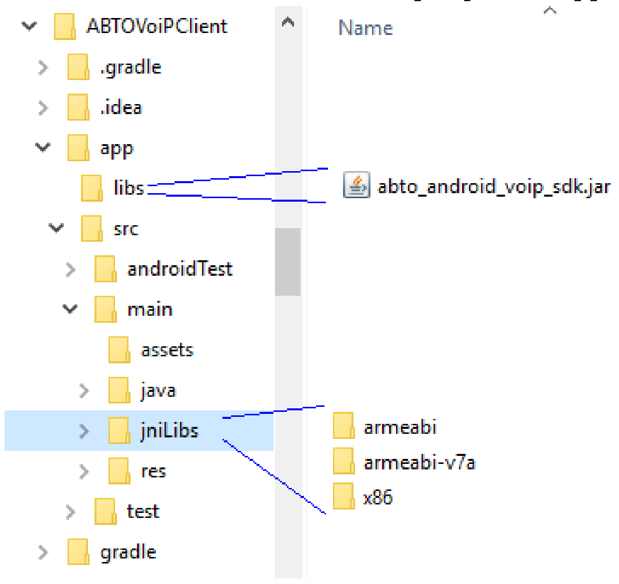
1. Copy libraries

a/ Copy *.jar file:

```
<SDK>\libs\abto_android_voip_sdk.jar
             to <project>\app\libs\abto_android_voip_sdk.jar
```

b/ Copy *.so files:

```
<SDK>\libs\armeabi     to project>\app\src\main\jniLibs
<SDK>\libs\armeabi-v7a to project>\app\src\main\jniLibs
<SDK>\libs\x86         to project>\app\src\main\jniLibs
```



2. Edit manifest file

Full version of manifest file you can see in example app “VoipTestStudio”, provided with SDK.

a/ Add permission

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.USE_SIP"/>
<uses-permission android:name="android.permission.CONFIGURE_SIP"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/><!--android 4-->
<uses-permission android:name="android.permission.WRITE_SETTINGS"/><!--android 4-->
```

b/ Add features

```
<uses-feature android:name="android.hardware.wifi"   android:required="false"/>
<uses-feature android:name="android.hardware.microphone"  android:required="true"/>
<uses-feature android:name="android.hardware.touchscreen" android:required="false"/>
<uses-feature android:name="android.hardware.bluetooth"   android:required="false"/>
<uses-feature android:name="android.hardware.screen.portrait" android:required="false"/>
<uses-feature android:name="android.hardware.camera"     android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-feature android:name="android.hardware.camera.flash" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front" android:required="false"/>
```

c/ Edit ‘application’ tag

Is required to use application class provided by SDK or own class extended from this one

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

Or

```
<application
      android:name=".MyApp"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name">

public class MyApp extends AbtoApplication {
    private long accountId;
```

d/ Add SDK service

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

e/ Add incoming call service

This service is required to handle incoming calls when app is in background and to prevent app process from closing (system can close all apps activities, but app continues to work and keep registration on server).
When incoming call received – service displays incoming call activity.

```
<service
      android:name=".IncomingCallService"
      android:enabled="true">
</service>
```

3. Modify build.gradle (Module: app)

Add line:

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
```

It’s required to build ‘abto_android_voip_sdk.jar’, copied on step 1.a

4. Add source code

a/ Configure and initialize phone instance

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

abtoPhone.setInitializeListener(this);

//configure codec
AbtoPhoneCfg config = abtoPhone.getConfig();
config.setCodecPriority(Codec.G729, (short) 250);
config.setCodecPriority(Codec.PCMU, (short) 200);
config.setCodecPriority(Codec.GSM, (short) 150);
config.setCodecPriority(Codec.PCMA, (short) 100);
config.setCodecPriority(Codec.speex_16000, (short) 50);
```

```
//Set port
config.setSipPort(0);//when app set ‘0’ - SDK will bind to random port number

//Set STUN server
config.setSTUNEnabled(true);
config.setSTUNServer(“stun.l.google.com:19302”);

//Set timeouts
config.setRegisterTimeout(5000);//how long SDK has to wait answer on ‘REGISTER’
config.setHangupTimeout(3000);//how long SDK has to wait answer on ‘BYE’


//Enable logs - SDK will store log on device in folder sdcard/package_name/logs
Log.setLogLevel(5);
Log.setUseFile(true);

//Set listener
abtoPhone.setInitializeListener(this);

//Start initializing - !has to be invoked only once, when  app started!
abtoPhone.initialize();
```

SDK invokes this method few times, during initialization:

```
@Override
public void onInitializeState(OnInitializeListener.InitializeState state, String message)
{
    switch (state) {
        case START:
        case INFO:
        case WARNING: break;
        case FAIL:              <- display error message here

        case SUCCESS:           <- switch to next activity or allow user to enter
registration credentials
```

b/ Register phone

To start registration invoke following code:

```
//Set listener
abtoPhone.setRegistrationStateListener(this);

// Add account
long accId = abtoPhone.getConfig().addAccount(domain, proxy, user, password, null, null,
300, true);

//Send register request
try {
    abtoPhone.register();
} catch (RemoteException e) {
    e.printStackTrace();
}
```

Then display some progress indicator and wait notification from SDK.

```
public void onRegistrationFailed(long accId, int statusCode, String statusText)
{
//SDK invokes this method when registration failed for some reason
//Display error message using method arguments:
    fail.setMessage(statusCode + " - " + statusText);
}
```

```
public void onRegistered(long accId)
{
//SDK invokes this method when registration was successful (server returned "200 OK")
//SDK invokes this method periodically (it updates registration using 'RegExpire' interval
- 6th argument of method 'addAccount')
}
```

c/ Start call and handle call events

To start new call invoke method:

```
try {
    abtoPhone.startCall(sipNumber, abtoPhone.getCurrentAccountId());//audio only call
//or
    abtoPhone.startVideoCall(sipNumber, abtoPhone.getCurrentAccountId());//video call

} catch (RemoteException e) {
    e.printStackTrace();
}
```

This code initiates call – prepares and sends 'SIP INVITE' request to remote side.

To handle call state events add handlers:

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

phone.setRemoteAlertingListener(this);
phone.setCallConnectedListener(this);
phone.setCallDisconnectedListener(this);
phone.setOnCallHeldListener(this);
phone.setToneReceivedListener(this);


public void onRemoteAlerting(long accId, int statusCode)
{
   //SDK invokes this method when received message from remote side (100, 180, 183..)

   switch (statusCode) {
   case TRYING:  statusText = "Trying";   break;
   case RINGING: statusText = "Ringing";  break;
   case SESSION_PROGRESS:   statusText = "Session in progress";    break;
   }
}


public void onCallDisconnected(String remoteContact, int callId)
{
//SDK invokes this method when:
 - Outgoing call was rejected on remote side
 - Successfully established call was cleared
 - SDK can't dial entered phone number (timeout)
}

@Override
public void onCallConnected(String remoteContact)
{
//SDK invokes this method call established
}

@Override
public void onCallHeld(HoldState state)
{
//SDK invokes this method call put on hold
}

@Override
public void onToneReceived(char tone)
{
//SDK invokes this method when received DTMF from remote side
}
```

To display captured and received video invoke method ‘`setVideoWindows`’:

```
LinearLayout outParrent = (LinearLayout) this.findViewById(R.id.local_video_parent);
LinearLayout inParrent = (LinearLayout)this.findViewById(R.id.remote_video_parent);
phone.setVideoWindows(outParrent, inParrent);
```

d/ Break established call

To hang-up established call or cancel initiated call invoke method:

```
try {
   phone.hangUp();
} catch (RemoteException e) {
   Log.e(THIS_FILE, e.getMessage());
}
```

Call is completed/canceled when received ‘onCallDisconected’.

e/ Receive incoming call

To receive incoming calls add listener

```
abtoPhone.setIncomingCallListener(this);


public void OnIncomingCall(String remoteContact, long arg1)
{
//SDK invokes this method when received incoming call
//To answer call with video app has to invoke: phone.answerCall(200, true);
//To answer call without video app has to invoke: phone.answerCall(200, false);
//To reject this call app has to invoke: phone.rejectCall();
}
```

f/ Unregister phone instance

To unregistered phone instance invoke code:

```
try {
    abtoPhone.unregister();
} catch (RemoteException e) {
    e.printStackTrace();
    dialog.dismiss();
}

@Override
public void onUnRegistered(long arg0)
{
//SDK invokes this method when registration was successfully removed
}
```

g/ Stop phone instance

To stop phone instance invoke code:

```
try {
    //Stop incoming call service
    Intent intent = new Intent(RegisterActivity.this, IncomingCallService.class);
    stopService(intent);

    //Destroy phone
    if(abtoPhone.isActive())         abtoPhone.destroy();

} catch (RemoteException e) {
    e.printStackTrace();
}
```

5. Build app