Android SIP-Native Call Integration for AJVoIP

The AJVoIP Android SIP SDK can be used to add SIP call capabilities for the native Android dialer.
This feature allows to capture outgoing calls initiated from the android native dialer and you can ask the user if he or she wants to use SIP or mobile (the phone operator network) for the call(s). The native call integration can be a useful and convenient feature for endusers who make a lot's of SIP call from their smartphone.

This feature is not built into the SIP SDK because it requires some user interface element which have to be implemented anyway by the developer, however we have created this document to guide you trough the process.

Native call integration for your Android SIP application can be easily implemented following the below provided simple example.

We can break down this task into two main steps:

1. Register an android BroadcastReceiver to "catch" the outgoing call.
2. Create an AlertDialog style Activity AlertDialog where the user can choose between SIP or mobile

Create and register BroadcastReceiver

We are going to create a new Java class called CallReceiver which extends BroadcastReceiver class and implement the onReceive() method to catch the outgoing call.

```
public class CallReceiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, final Intent intent)
  {
 // catch incoming calls and handle
    // filter intent for outgoing calls
    if (intent.getAction().equals("android.intent.action.NEW_OUTGOING_CALL"))
    {
      // get the called number from intent
      String phoneNumber = intent.getExtras().getString( "android.intent.extra.PHONE_NUMBER");

      if (phoneNumber.equals("911") || phoneNumber.equals("112")) // exclude emergency numbers
      {
        return;
      }

       // start the AlertDialog activity, so the user can choose
       // the implementation will be described below
      Intent intentDialog = new Intent(getAppContext(), CallChooser.class);
      intentDialog.putExtra("callednumber", phoneNumber.trim());
      intentDialog.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
      intentDialog.addFlags(Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);
      getAppContext().startActivity(intentDialog);

      setResultData(null);
    }
  }
}
```

Add the following lines in your AndroidManifest.xml:

- Add the following permissions outside of the <application> tag:

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

- Register the your broadcast receiver (CallReceiver)

```
<receiver android:name=".CallReceiver">
          <intent-filter>
              <action android:name="android.intent.action.PHONE_STATE" />
          </intent-filter>
       <intent-filter android:priority="0">
```

```
            <action    android:name="android.intent.action.NEW_OUTGOING_CALL" />
        </intent-filter>
    </receiver>
```

- Also register your Activity, CallChooser, which we will create for presenting the user the option to choose

```
<activity android:name=".CallChooser" android:theme="@android:style/Theme.Dialog" />
```

Create dialog style Activity

We are going to create a new Java class called CallChooser which extends Activity. This activity will be started from CallReceiver onReceive() methond when the user makes an outgoing call. Let's create the **call_chooser.xml** which defines the user interface content of the activity, like this:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <LinearLayout android:orientation="horizontal"
       android:layout_width="fill_parent" android:layout_height="wrap_content"
       android:padding="10dp">

      <TextView android:id="@+id/call_dialog_title"
         android:layout_width="fill_parent" android:layout_height="wrap_content"
           android:layout_weight="1" android:textSize="18dp" />
   </LinearLayout>

   <LinearLayout android:id="@+id/layout_call_android"
      android:orientation="vertical" android:layout_height="wrap_content"
      android:layout_width="fill_parent" android:paddingTop="3dp"
      android:paddingBottom="3dp" android:paddingLeft="15dp"
      android:clickable="true" android:focusable="true">

      <TextView android:id="@+id/android_caller"
         android:layout_width="fill_parent" android:layout_height="wrap_content"
         android:layout_weight="1" android:textSize="18dp"
         android:text="Use AJVoIP"/>
   </LinearLayout>

   <LinearLayout android:id="@+id/layout_call_native"
      android:orientation="vertical" android:layout_height="wrap_content"
      android:layout_width="fill_parent" android:paddingTop="3dp"
      android:paddingBottom="3dp" android:paddingLeft="15dp"
      android:clickable="true" android:focusable="true">

      <TextView android:id="@+id/native_caller"
         android:layout_width="fill_parent" android:layout_height="wrap_content"
         android:layout_weight="1" android:textSize="18dp"
         android:text="Use mobile"/>
   </LinearLayout>
</LinearLayout>
```

Now let's add the code for in CallChooser.java:

```
public class CallChooser extends Activity
{
   private String calledNumber = "";

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.call_chooser);

       LinearLayout androidLayout = (LinearLayout) findViewById(R.id.layout call android);
      LinearLayout nativeLayout = (LinearLayout) findViewById(R.id.layout call native);
      TextView title = (TextView) findViewById(R.id.call_dialog_title);

      Bundle extras = getIntent().getExtras();
        if (extras != null)
        {
           calledNumber = extras.getString("callednumber");
        }

        title.setText("Call " + calledNumber);
```

```
      androidLayout.setOnClickListener(new View.OnClickListener()
        {
           public void onClick(View v)
         {
            ajvoipinstance.Call(-1, calledNumber);
            finish();
         }
        });

      nativeLayout.setOnClickListener(new View.OnClickListener()
        {
           public void onClick(View v)
         {
                // continue with native call
                String uri = "tel:" + calledNumber;
                Intent intentNative = new Intent(Intent.ACTION_CALL);
                intentNative.setData(Uri.parse(uri));
                startActivity(intentNative);
         }
        });
    }
}
```

This concludes our simple implementation example of native call integration.
It goes without saying, that these are just basic examples which can be further customized or improved after your needs. For example you can present this option for the user only once and save his/hers preferred choice.

Handle native call on VoIP busy

You might choose to reject incoming native calls (GSM/LTE by the mobile operator) if there is already a VoIP call in progress. For this we need to "capture" the incoming native call. This can be achieved by registering an android BroadcastReceiver. We are going to create a new Java class called CallReceiver which extends BroadcastReceiver class and implement the onReceive() method to catch the incoming call:

```
public class CallReceiver extends BroadcastReceiver
{
   @Override
   public void onReceive(Context context, final Intent intent)
   {
      // filter intent for phone states (including incoming calls)
      if (intent.getAction().equals("android.intent.action.PHONE_STATE"))
      {
          String state = intent.getStringExtra(TelephonyManager.EXTRA_STATE);

          // this means that we have a native incoming call
          if (state.equals(TelephonyManager.EXTRA_STATE_RINGING))
          {
              // we can get the incoming caller number
              String incomingNumber = intent.getStringExtra(TelephonyManager.EXTRA_INCOMING_NUMBER);

               // you can check here, if currently there is a VoIP call in progress
               // and choose how to handle it

               // For example, you could reject the native call
               // Below you will find an example, of how to reject the native call
          }
          setResultData(null);

      }
   }
 }


// Example of rejecting an icoming native call

public void RejectNativeCall(Context context)
{
   try {
      // Get the TelephonyManager
      TelephonyManager telephonyManager = (TelephonyManager)
context.getSystemService(Context.TELEPHONY_SERVICE);
      Class classTelephony = Class.forName(telephonyManager.getClass().getName());
```

```
        Method methodGetITelephony = classTelephony.getDeclaredMethod("getITelephony");

        // Ignore that the method is supposed to be private
        methodGetITelephony.setAccessible(true);

        // Invoke getITelephony() to get the ITelephony interface
        Object telephonyInterface = methodGetITelephony.invoke(telephonyManager);

        // Get the endCall method from ITelephony
        Class telephonyInterfaceClass = Class.forName(telephonyInterface.getClass().getName());
        Method methodEndCall = telephonyInterfaceClass.getDeclaredMethod("endCall");

        // Invoke endCall()
        methodEndCall.invoke(telephonyInterface);

    }catch(Throwable e){ Log.e("AJVOIP", "ERROR,Failed to end native call"); }
}
```

Add the following lines in your AndroidManifest.xml:

- Add the following permissions outside of the <application> tag:
  ```
  <uses-permission android:name="android.permission.CALL_PHONE"/>
   <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
  ```

- Register the your broadcast receiver (CallReceiver)

```
<receiver android:name=".CallReceiver">
      <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" />
      </intent-filter>
    <intent-filter android:priority="0">
      <action  android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>
```

Manage Native call history

The following examples will show how to get all entries from native call history and how to insert entries into.

Note: for the below examples to work, you will need READ_CALL_LOG and WRITE_CALL_LOG permissions.

```
        public void GetCallHistory()
        {
            Uri uriAllCalls = Uri.parse("content://call_log/calls");
            ContentResolver cr = getContentResolver();
            Cursor cur = cr.query(uriAllCalls, null, null, null, null);

            if (cur != null && cur.getCount() > 0)
            {
                while (cur.moveToNext())
                {
                    // below we will have the name/number/time/type of one native call history entry
                    String number = cur.getString(cur.getColumnIndex(CallLog.Calls.NUMBER));
                    String name = cur.getString(cur.getColumnIndex(CallLog.Calls.CACHED_NAME));
                    // time of call in milliseconds since epoch
                    long timeofcall = cur.getLong(cur.getColumnIndex(CallLog.Calls.DATE));
                    // main types can be: CallLog.Calls.OUTGOING_TYPE, CallLog.Calls.INCOMING_TYPE, CallLog.Calls.MISSED_TYPE
                    int type = cur.getInt(cur.getColumnIndex(CallLog.Calls.TYPE));
                }
            }

            try{ if(cur != null){ cur.close(); } cur = null; }catch(Throwable e){ }
        }
```

```
// insert an entry into native call history

public void InsertCallHistoryEntry(String number, String name, long timeofcall, long callduration, int type)
{
   // timeofcall: timestamp when the call was made in milliseconds since epoch
   // callduration: duration of a connected call in milliseconds
   // type: the type of the call can be: CallLog.Calls.OUTGOING_TYPE, CallLog.Calls.INCOMING_TYPE,
CallLog.Calls.MISSED_TYPE
   ContentResolver cr = getContentResolver();

   ContentValues values = new ContentValues();
   values.put(CallLog.Calls.NUMBER, number);
   values.put(CallLog.Calls.DATE, timeofcall);
   values.put(CallLog.Calls.DURATION, callduration);
   values.put(CallLog.Calls.TYPE, type);
   values.put(CallLog.Calls.NEW, 1);
   values.put(CallLog.Calls.CACHED_NAME, name);
   values.put(CallLog.Calls.CACHED_NUMBER_TYPE, 0);
   values.put(CallLog.Calls.CACHED_NUMBER_LABEL, "");

   cr.insert(CallLog.Calls.CONTENT_URI, values);
}
```

Get native SIM phone number

Not strictly related to call integration, but related to phone integration so we list it here for your convenience. For example you might wish to use the user phone number as it's SIP user name during new registration thus you might try to read it from the device (and ask the user if this is not possible).

This example will return the android device SIM phone number if the number is stored on the users SIM card and the android OS has access to it, otherwise it will return an empty string. Requires READ_PHONE_STATE permission.

```
public String GetSimNumber()
{
   TelephonyManager tMgr =(TelephonyManager)this.getSystemService(Context.TELEPHONY_SERVICE);
   String simNumber = tMgr.getLine1Number();

   if (simNumber == null) simNumber = "";
   if (simNumber.length() > 0)
   {
      simNumber = simNumber.trim();
      simNumber = simNumber.replace("-", "");
      simNumber = simNumber.replace("+", "");

      return simNumber;
   }
   return "";
}
```

Additional integration

It is up to your needs to add any other integration after your needs for your SIP app with the Android OS such offering services for other application or consuming services offered by the OS or by other applications.
A few ideas:

- run your app as a service (described in the Android SIP SDK documentation "Run as Service" FAQ point)
- start other activities
- interacting with other apps
- handling app links
- create shortcuts
- create widgets
- contact management (described in the Android SIP SDK documentation "Contact Management" FAQ point)
- send SMS (described in the Android SIP SDK documentation "How to send SMS" FAQ point)
- integrate with any existing API (HTTP API requests)
- integrate any existing web page into your phone by using WebView
- consult the AJVoIP documentation and the Android developer guide for more help or ideas