Application Note

# WebRTC App Note

**A quick overview of this set of Application Notes**

ReconOS combined with Recon Jet Pro provides a powerful Operating System for Enterprises. This guide offers a quick insight into the use of WebRTC to create a video/audio chat app.

Version 1.0

# Table of Contents

## Introduction

This is a brief overview of the information and tools you'll need to develop a WebRTC video/audio chat app on your Android powered Recon Jet device. The app will allow you to connect with another user over the internet and view his video/audio stream on your device. This Application Note expects that you have read "AN002 – Developing your first Recon App"

## Sample Code Location

*/Sample Code/WebRTCDemo*

## Tutorial

These instructions will walk you through the process of creating an app key on PubNub. You'll learn how to create a basic UI that will help facilitate the function of calling another user. You'll also learn how to add WebRTC functionality to your project and implement a simple video/audio chat app.

## Creating App Keys on PubNub

In this part of the tutorial, we'll show you how to create an app key on PubNub. For a complete implementation of this tutorial, see the source code of our <LINK> WebRTCDemo <LINK> sample app. However, in order for WebRTCDemo to work, you will need to complete this first part of creating and adding a PubNub key to your app.

First, create an account on www.pubnub.com.

Once you've successfully created an account and logged in (to the main dashboard page https://admin.pubnub.com), click on button called 'New App +' (located on the top bar). Type in a name for your app and click create.

Now, clicking on the newly created app, you will see a box called 'Demo Keyset'. This box contains both a PUBLISH KEY and a SUBSCIBE KEY. They should resemble the following, respectively:

**pub-c-4745a4f2-b84e-45ec-bac4-3c34bc0a1c12**

**sub-c-2ad24e68-7f2c-34e6-ada6-02ee2ddab7fe**

You will need these keys in order to access the PubNub API in our implementation. PubNub will allow our app to connect two users for a WebRTC video/audio chat from anywhere with a solid internet connection!

## Adding Dependencies

Like in some previous sample app tutorials, we need to add a few dependencies to our project. Add the following lines to your build.gradle (Module: app) files dependencies section:

```
compile ’io.pristine:libjingle:9127@aar’
compile ’com.pubnub:pubnub-android:3.7.4’
compile ’me.kevingleason:pnwebrtc:1.0.6@aar’
```

We will go over the use of these packages as we go through this tutorial.

## Connecting Users With PubNub

In this section we will go over how to connect two users using the Pubnub API. First we need to import the Pubnub packages:

```
import com.pubnub.api.Callback;
import com.pubnub.api.Pubnub;
import com.pubnub.api.PubnubError;
import com.pubnub.api.PubnubException;
```

Next within our Constants class, we declare a couple public static variables:

```
final String STDBY_SUFFIX = ”-stdby”;
final String PUB_KEY      = ”pub-c-4745a4f2-b84e-45ec-bac4-3c34bc0a1c12”;
final String SUB_KEY      = ”sub-c-2ad24e68-7f2c-34e6-ada6-02ee2ddab7fe”;
final String USER_NAME    = ”user_name”;
final String CALL_USER    = ”call_user”;
```

We make it in a separate class because these variables will be referenced from outside of our main class (main layout).

Make sure that you use your own app keys (PUBLISH and SUBSCRIBE) from Step 1, for the above PUB_KEY and SUB_KEY respectively.

Heading back to our Main class, in order to create a call between two users, we need to have one part of our app dedicated to listening for an incoming call. In order to do this we need to create an ID (UUID) for our device and then register a channel for listening to incoming calls, using the Pubnub app.

We do this by adding the next section of code:

```
public void initPubNub(String nameMe, String nameYou)
{
    this.userMe  = nameMe;
    this.userYou = nameYou;

    String stdbyChannel = this.userMe + Constants.STDBY_SUFFIX;
    this.mPubNub = new Pubnub(Constants.PUB_KEY, Constants.SUB_KEY);
    this.mPubNub.setUUID(this.userMe);
    try
    {
        // Subscribe our channel on PubNub and specify our callback handlers
        this.mPubNub.subscribe(stdbyChannel, new Callback()
        {
            @Override
            public void successCallback(String channel, Object message)
            {
                try{
                    JSONObject jsonMsg = (JSONObject) message;
                    if(!jsonMsg.has(Constants.CALL_USER)){ return; }
                    userYou = jsonMsg.getString(Constants.CALL_USER);
                    launchViewActivity();
                }catch(JSONException e){ e.printStackTrace(); }
            }

            @Override
            public void connectCallback(String channel, Object message){ }
        });
    }
    catch (PubnubException e){ e.printStackTrace(); }
}
```

In the above section you can see that we register a PubNub channel with our name and a postfix string. Then we wait for a successful callback, at which point we pull the other users name out of the sent JSON object and launch our view activity.

Now, we also need the ability to call a user, who is listening on our Pubnub channel.The next bit of functionality gives our app the ability to make a call to another user with a specific name (only 2 user names are used in our sample app). Notice that we create a JSON Object and add the user name for who we are attempting to call (this user name is grabbed in the above code):

```
public void initializeCall(final String callNum)
{
    final String callNumStdBy = callNum + Constants.STDBY_SUFFIX;
    try
    {
        // Create a JSON object with call number (other users name)
        JSONObject jsonCall = new JSONObject();
        jsonCall.put(Constants.CALL_USER, callNum);

        // Publish our call and specify our callback handlers
        mPubNub.publish(callNumStdBy, jsonCall, new Callback()
        {
            @Override
            public void successCallback(String channel, Object message)
            {
                launchViewActivity();
            }

            @Override
            public void errorCallback(String channel, PubnubError error){ }
        });
    }
    catch(JSONException e){ e.printStackTrace(); }
}
```

Upon successfully making or receiving a call, we launch our view activity. This activity is going to establish the WebRTC connection and display it on both devices screens. We can launch this activity via the following:

```
void launchViewActivity()
{
    Intent intent = new Intent(MainActivity.this, ViewActivity.class);
    intent.putExtra(Constants.USER_NAME, userMe);
    intent.putExtra(Constants.CALL_USER, userYou);
    startActivity(intent);
}
```

## Initializing WebRTC

In this section we will go over how to run up WebRTC.

First we need to make sure our ViewActivity is using a layout with a GLSurfaceView. In our activity_view.xml layout file, we add the following:

```
...
<android.opengl.GLSurfaceView
   android:id="@+id/glview"
   android:layout_width="match_parent"
   android:layout_height="match_parent" />
...
```

Now back to our ViewAcitivity class, we need to import some WebRTC and other packages:

```
import org.webrtc.AudioSource;
import org.webrtc.AudioTrack;
import org.webrtc.MediaConstraints;
import org.webrtc.MediaStream;
import org.webrtc.PeerConnectionFactory;
import org.webrtc.VideoCapturerAndroid;
import org.webrtc.VideoRenderer;
import org.webrtc.VideoRendererGui;
import org.webrtc.VideoSource;
import org.webrtc.VideoTrack;

import me.kevingleason.pnwebrtc.PnPeer;
import me.kevingleason.pnwebrtc.PnRTCClient;
import me.kevingleason.pnwebrtc.PnRTCListener;
```

We then need to declare our variables and initialize them within our onCreate(...) method:

```

public static final String VIDEO_TRACK_ID       = "videoPN";
public static final String AUDIO_TRACK_ID       = "audioPN";
public static final String LOCAL_MEDIA_STREAM_ID = "localStreamPN";

private TextView mTextLog;
private GLSurfaceView mGLView;

private PnRTCClient pnRTCClient;
private VideoSource localVideoSource;
private VideoRenderer.Callbacks localRender;
private VideoRenderer.Callbacks remoteRender;
private String username;

@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.activity_view);

    // Make sure our username was passed in (via Intent)
    Bundle extras = getIntent().getExtras();
    if(extras == null || !extras.containsKey(Constants.USER_NAME))
    {
        finish();
        return;
    }
    this.username = extras.getString(Constants.USER_NAME, "");
    logText("My Username:: " + this.username);

    mTextLog = (TextView)findViewById(R.id.textLog);
    mGLView  = (GLSurfaceView)findViewById(R.id.glview);
    initWebRTC(extras);
}
```

After this we can simply initialize our WebRTC objects.

This setup is relatively straight forward and requires a number of different custom options. In our sample app it is laid out as the following:

```
void initWebRTC(Bundle extras)
{
    try
    {
        // Setup variables
        final Context c             = this;
        final boolean initAudio     = true;
        final boolean initVideo     = true;
        final boolean vidCodecHWAccel = true;
        final Object renderEGLContext = null;

        boolean success = PeerConnectionFactory.initializeAndroidGlobals(c, initAudio, initVideo,
vidCodecHWAccel, renderEGLContext);

        if(!success)
        {
            finish();
            throw new java.lang.Exception("PeerConnectionFactory - Failed to Initialize Android
Globals.");
        }
        logText("[PeerConnectionFactory.initializeAndroidGlobals(...)]");

        PeerConnectionFactory peerConnectionFactory = new PeerConnectionFactory();
        logText("[new PeerConnectionFactory()]");


        int deviceCount = VideoCapturerAndroid.getDeviceCount();
        logText("Capture Device Count: " + deviceCount);
        String frontFaceDeviceString      = VideoCapturerAndroid.getNameOfFrontFacingDevice();
        String backFaceDeviceString       = VideoCapturerAndroid.getNameOfBackFacingDevice();
        VideoCapturerAndroid videoCapturer =
VideoCapturerAndroid.create(VideoCapturerAndroid.getDeviceName(deviceCount-1));

        MediaConstraints videoConstraints = new MediaConstraints();
        videoConstraints.mandatory.add(new MediaConstraints.KeyValuePair("maxWidth",  "428"));
        videoConstraints.mandatory.add(new MediaConstraints.KeyValuePair("maxHeight", "240"));
        localVideoSource          = peerConnectionFactory.createVideoSource(videoCapturer,
videoConstraints);
        VideoTrack localVideoTrack = peerConnectionFactory.createVideoTrack(VIDEO_TRACK_ID,
localVideoSource);
        logText("[peerConnectionFactory.createVideoTrack(...)]");

        MediaConstraints audioConstraints = new MediaConstraints();
        AudioSource audioSource    = peerConnectionFactory.createAudioSource(audioConstraints);
        AudioTrack localAudioTrack = peerConnectionFactory.createAudioTrack(AUDIO_TRACK_ID,
audioSource);
        logText("[peerConnectionFactory.createAudioTrack(...)]");

        // Then we set that view, and pass a Runnable to run once the surface is ready
        VideoRendererGui.setView(mGLView, null);
        logText("[VideoRendererGui.setView(...)]");

        // Now that VideoRendererGui is ready, we can get our VideoRenderer. IN THIS ORDER.
Effects which is on top or bottom
        remoteRender = VideoRendererGui.create(0, 0, 100, 100,
VideoRendererGui.ScalingType.SCALE_ASPECT_FILL, false);
        localRender  = VideoRendererGui.create(0, 0, 100, 100,
VideoRendererGui.ScalingType.SCALE_ASPECT_FILL, false);

        // We start out with an empty MediaStream object, created with help from our
PeerConnectionFactory
        MediaStream mediaStream =
peerConnectionFactory.createLocalMediaStream(LOCAL_MEDIA_STREAM_ID);
        mediaStream.addTrack(localVideoTrack);
        mediaStream.addTrack(localAudioTrack);

        connectPubNubWebRTCClient(extras, mediaStream);
```

```
   }
   catch(java.lang.Exception e)
   {
      logText("ERROR - [FAILED TO INITIALIZE!] ");
      e.printStackTrace();
   }
}
```

Once we initialize our WebRTC objects, we are ready to hook up our media stream to our PubNub-WebRTC client. We do this with the following:

```
void connectPubNubWebRTCClient(Bundle extras, MediaStream mediaStream)
   {
      this.pnRTCClient = new PnRTCClient(Constants.PUB_KEY, Constants.SUB_KEY,
this.username);
      // First attach the RTC Listener so that callback events will be triggered
      this.pnRTCClient.attachRTCListener(new MyRTCListener());
      this.pnRTCClient.attachLocalMediaStream(mediaStream);
      // Listen on a channel. This is your "phone number," also set the max chat users.
      this.pnRTCClient.listenOn(this.username);
      this.pnRTCClient.setMaxConnections(1);

      // If Constants.CALL_USER is in the intent extras, auto-connect them.
      if(extras.containsKey(Constants.CALL_USER))
      {
         String callUser = extras.getString(Constants.CALL_USER, "");
         pnRTCClient.connect(callUser);
      }
   }
```

Finally we need to initialize some callbacks for when our client has successfully established a connection using our WebRTC media streams:

```
private class MyRTCListener extends PnRTCListener
   {
      @Override
      public void onLocalStream(final MediaStream localStream)
      {
         ViewActivity.this.runOnUiThread(new Runnable()
         {
            @Override
            public void run()
            {
               logText("MyRTCListenet - onLocalStream.run()");
               if(localStream.videoTracks.size() == 0){ return; }
               localStream.videoTracks.get(0).addRenderer(new VideoRenderer(localRender));
            }
         });
      }
```

```
        @Override
        public void onAddRemoteStream(final MediaStream remoteStream, final PnPeer peer)
        {
            ViewActivity.this.runOnUiThread(new Runnable()
            {
                @Override
                public void run()
                {
                    logText("MyRTCListenet - onAddRemoteStream.run()");
                    Toast.makeText(ViewActivity.this, "Connected to " + peer.getId(),
Toast.LENGTH_SHORT).show();
                    try
                    {
                        if(remoteStream.videoTracks.size() == 0){ return; }
                        remoteStream.videoTracks.get(0).addRenderer(new
VideoRenderer(remoteRender));
                        VideoRendererGui.update(remoteRender, 0, 0, 100, 100,
VideoRendererGui.ScalingType.SCALE_ASPECT_FILL, false);
                        VideoRendererGui.update(localRender, 72, 72, 25, 25,
VideoRendererGui.ScalingType.SCALE_ASPECT_FIT, true);
                    }
                    catch (Exception e){ e.printStackTrace(); }
                }
            });
        }

        @Override
        public void onPeerConnectionClosed(PnPeer peer)
        {
            logText("MyRTCListenet - onPeerConnectionClosed.run()");
            finish();
        }
    }
```

Please refer to the sample app for a better understanding of all these components.