**Main activity**

package com.vvv.wificall.activity;

import com.vvv.wificall.activity.R;

import com.vvv.wificall.activity.task.StartNetworkTask;

import com.vvv.wificall.event.CallbackEvent;

import android.app.Activity;

import android.content.Context;

import android.content.Intent;

import android.os.Bundle;

import android.os.PowerManager;

import android.os.PowerManager.WakeLock;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.EditText;

public class MainActivity extends Activity

{

private CallbackEvent startNetworkCallback = null;

private StartNetworkTask startNetworkTask = null;

private WakeLock wakeLock = null;

public static EditText StatusField = null;

public static EditText NameField = null;

public static Button TransmitButton = null;

public static Button ConnectButton = null;

public static Button ExitButton = null;

@Override

public void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

PowerManager powerManager = (PowerManager)getSystemService(Context.POWER\_SERVICE);

wakeLock = powerManager.newWakeLock(PowerManager.FULL\_WAKE\_LOCK, "DoNotDimScreen");

StatusField = (EditText)findViewById(R.id.editText1);

NameField = (EditText)findViewById(R.id.editText2);

TransmitButton = (Button)findViewById(R.id.button2);

TransmitButton.setTag(false);

TransmitButton.setOnClickListener(new OnClickListener()

{

public void onClick(View arg0)

{

TransmitButton.setTag(!(Boolean)TransmitButton.getTag());

if ((Boolean)TransmitButton.getTag())

TransmitButton.setText("End Call");

else

TransmitButton.setText("Call");

}

});

ConnectButton = (Button)findViewById(R.id.button1);

ConnectButton.setTag(false);

ConnectButton.setOnClickListener(new OnClickListener()

{

public void onClick(View arg0)

{

if ((Boolean)ConnectButton.getTag())

{

startNetworkTask.stopNotifier.stop();

ConnectButton.setText("Disconnecting...");

ConnectButton.setEnabled(false);

TransmitButton.setEnabled(false);

if ((Boolean)TransmitButton.getTag())

TransmitButton.performClick();

}

else

{

startNetworkTask = new StartNetworkTask(MainActivity.this, startNetworkCallback);

startNetworkTask.execute(null, null);

ConnectButton.setText("Cancel");

ConnectButton.setTag(true);

}

}

});

ExitButton = (Button)findViewById(R.id.button3);

ExitButton.setOnClickListener(new OnClickListener()

{

public void onClick(View arg0)

{

System.exit(0);

}

});

startNetworkCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

TransmitButton.setEnabled(false);

ConnectButton.setTag(false);

ConnectButton.setText("Search/Create");

ConnectButton.setEnabled(true);

}

};

}

@Override

protected void onPause()

{

super.onPause();

wakeLock.release();

}

@Override

protected void onResume()

{

super.onResume();

wakeLock.acquire();

}

@Override

public boolean onCreateOptionsMenu(Menu menu)

{

menu.add(0, 0, 0, "Preferences");

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item)

{

switch (item.getItemId())

{

case 0:

{

Intent intent = new Intent(MainActivity.this, PreferencesActivity.class);

startActivity(intent);

return true;

}

}

return false;

}

}

**Preference Activity**

package com.vvv.wificall.activity;

import com.vvv.wificall.activity.R;

import android.os.Bundle;

import android.preference.PreferenceActivity;

public class PreferencesActivity extends PreferenceActivity

{

@Override

protected void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

addPreferencesFromResource(R.xml.preferences);

}

}

**Start network Activity**

package com.vvv.wificall.activity.task;

import com.vvv.wificall.activity.MainActivity;

import com.vvv.wificall.event.CallbackEvent;

import com.vvv.wificall.event.ObjectHolder;

import com.vvv.wificall.event.StopEvent;

import com.vvv.wificall.wifi.NetworkAdapter;

import com.vvv.wificall.wifi.SessionAdapter;

import com.vvv.wificall.wifi.TransmissionAdapter;

import com.vvv.wificall.wifi.SessionAdapter.SessionMessage;

import android.content.Context;

import android.media.AudioFormat;

import android.media.AudioManager;

import android.media.AudioRecord;

import android.media.AudioTrack;

import android.media.MediaRecorder.AudioSource;

import android.os.AsyncTask;

public class StartNetworkTask extends AsyncTask<Object, Object, Object>

{

private final static int SendPacketTimeout = 5000;

private final static int ReceivePacketTimeout = 500;

private final static int TransactionTimeout = 10000;

private final static int FastTimeSpan = 100;

public StopEvent stopNotifier = null;

private Context context = null;

private CallbackEvent callback = null;

private CallbackEvent statusCallback = null;

private CallbackEvent runtimeCallback = null;

private CallbackEvent interfaceCallback = null;

private CallbackEvent timeoutCallback = null;

private AudioTrack voice = null;

public StartNetworkTask(Context context, CallbackEvent callback)

{

super();

currently non-return for an application

this.context = context;

this.callback = callback;

stopNotifier = new StopEvent();

statusCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 0, arg });

}

};

runtimeCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 1, arg });

}

};

interfaceCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 2, arg });

}

};

timeoutCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 3, arg });

}

};

try

{

voice = new AudioTrack(AudioManager.STREAM\_MUSIC,

8000,

AudioFormat.CHANNEL\_CONFIGURATION\_MONO,

AudioFormat.ENCODING\_PCM\_16BIT,

TransmissionAdapter.MaxPacketSize - 2,

AudioTrack.MODE\_STREAM);

voice.setPlaybackRate(8000);

voice.play();

}

catch (Exception e)

{

e.printStackTrace();

}

}

@Override

protected void onPreExecute()

{

super.onPreExecute();

}

private String ByteArrayToString(byte[] bytes)

{

String result = "";

for (int i = 0; ((i < bytes.length) && (bytes[i] != 0)); i ++)

result += (char)bytes[i];

return result;

}

private void OnConnected()

{

byte[] inputBuffer = new byte[TransmissionAdapter.MaxPacketSize];

byte[] payloadBuffer = new byte[TransmissionAdapter.MaxPacketSize - 2];

statusCallback.onCallback("Waiting...");

int timeout = -1;

while (!stopNotifier.isStopped())

{

if ((Boolean)MainActivity.TransmitButton.getTag())

{

if (TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_TRANSMISSION\_BEGIN,

null),

stopNotifier, SendPacketTimeout, FastTimeSpan)

)

{

statusCallback.onCallback("Talking...");

AudioRecord record = null;

try

{

record = new AudioRecord(AudioSource.MIC,

8000,

AudioFormat.CHANNEL\_CONFIGURATION\_MONO,

AudioFormat.ENCODING\_PCM\_16BIT,

TransmissionAdapter.MaxPacketSize - 2);

record.startRecording();

}

catch (Exception e)

{

e.printStackTrace();

}

timeout = TransactionTimeout;

while (!stopNotifier.isStopped() && (Boolean)MainActivity.TransmitButton.getTag())

{

if (record != null)

record.read(payloadBuffer, 0, TransmissionAdapter.MaxPacketSize - 2);

if (TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_TRANSMISSION\_PAYLOAD,

payloadBuffer),

stopNotifier, SendPacketTimeout, FastTimeSpan))

timeout = TransactionTimeout;

else

{

timeout -= SendPacketTimeout;

if (timeout < 0)

{

timeout = -1;

timeoutCallback.onCallback(null);

break;

}

}

}

record.stop();

statusCallback.onCallback("Waiting...");

TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_TRANSMISSION\_END,

null),

null, SendPacketTimeout, FastTimeSpan);

}

}

else

{

if (TransmissionAdapter.ReceivePackets(inputBuffer, null, stopNotifier, ReceivePacketTimeout, FastTimeSpan))

{

switch (SessionAdapter.PacketDispatcher(inputBuffer, payloadBuffer))

{

case MSG\_TRANSMISSION\_BEGIN:

{

timeout = TransactionTimeout;

statusCallback.onCallback("Listening...");

interfaceCallback.onCallback(false);

break;

}

case MSG\_TRANSMISSION\_PAYLOAD:

{

timeout = TransactionTimeout;

voice.write(payloadBuffer, 0, payloadBuffer.length);

break;

}

case MSG\_TRANSMISSION\_END:

{

timeout = -1;

statusCallback.onCallback("Waiting...");

interfaceCallback.onCallback(true);

break;

}

case MSG\_DISCONNECT:

{

return;

}

}

}

if (timeout != -1)

{

if (timeout >= 0)

timeout -= ReceivePacketTimeout;

if (timeout < 0)

{

timeout = -1;

statusCallback.onCallback("Waiting...");

interfaceCallback.onCallback(true);

}

}

}

}

timeout = TransactionTimeout;

while (!TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_DISCONNECT,

null),

null, SendPacketTimeout, FastTimeSpan))

{

timeout -= SendPacketTimeout;

if (timeout < 0)

break;

}

}

@Override

protected Object doInBackground(Object... params)

{

byte[] inputBuffer = new byte[TransmissionAdapter.MaxPacketSize];

byte[] payloadBuffer = new byte[TransmissionAdapter.MaxPacketSize - 2];

if (NetworkAdapter.BeginNetworkScan(context, statusCallback, stopNotifier))

{

if (NetworkAdapter.BeginNetworkConnection(context, statusCallback, stopNotifier))

{

NetworkAdapter.SetOtherClientIP(NetworkAdapter.GetServerIP(context));

statusCallback.onCallback("Sending authorization request...");

if (TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_CONNECTION\_REQUEST,

NetworkAdapter.GetUserName(context).getBytes()),

stopNotifier, -1, -1)

)

{

statusCallback.onCallback("Receiving autorization response...");

if (TransmissionAdapter.ReceivePackets(inputBuffer, null, stopNotifier, -1, -1))

{

switch (SessionAdapter.PacketDispatcher(inputBuffer, payloadBuffer))

{

case MSG\_CONNECTION\_SUCCESS:

{

NetworkAdapter.SetOtherName(ByteArrayToString(payloadBuffer));

statusCallback.onCallback("Outcoming connection established.");

runtimeCallback.onCallback(true);

OnConnected();

runtimeCallback.onCallback(false);

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

NetworkAdapter.SetOtherName("---");

statusCallback.onCallback("Conection closed.");

break;

}

case MSG\_CONNECTION\_FAILURE:

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Authorization rejected by server.");

break;

}

default:

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Authorization failed.");

break;

}

}

}

else

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Server is not responding.");

}

}

else

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Network unreachable.");

}

}

else

{

statusCallback.onCallback("Connection failed.");

}

}

else if (!stopNotifier.isStopped())

{

if (NetworkAdapter.StartAccessPoint(context, statusCallback, stopNotifier))

{

boolean isUserConnected = false;

ObjectHolder<String> senderIPHolder = new ObjectHolder<String>();

while (!stopNotifier.isStopped() && !isUserConnected)

{

statusCallback.onCallback("Waiting for connections...");

if (TransmissionAdapter.ReceivePackets(inputBuffer, senderIPHolder, stopNotifier, -1, -1))

{

switch (SessionAdapter.PacketDispatcher(inputBuffer, payloadBuffer))

{

case MSG\_CONNECTION\_REQUEST:

{

NetworkAdapter.SetOtherClientIP(senderIPHolder.getObject());

statusCallback.onCallback("Sending authorization response...");

if (TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_CONNECTION\_SUCCESS,

NetworkAdapter.GetUserName(context).getBytes()),

stopNotifier, -1, -1)

)

{

isUserConnected = true;

}

break;

}

}

}

}

if (isUserConnected)

{

NetworkAdapter.SetOtherName(ByteArrayToString(payloadBuffer));

statusCallback.onCallback("Incoming connection established.");

runtimeCallback.onCallback(true);

OnConnected();

runtimeCallback.onCallback(false);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

NetworkAdapter.SetOtherName("---");

}

NetworkAdapter.StopAccessPoint(context, statusCallback, null);

statusCallback.onCallback("Conection closed.");

}

else

{

statusCallback.onCallback("Access Point not created.");

}

}

if (stopNotifier.isStopped())

statusCallback.onCallback("Idle.");

return 0;

}

@Override

protected void onProgressUpdate(Object... progress)

{

super.onProgressUpdate(progress);

switch ((Integer)progress[0])

{

case 0:

{

MainActivity.StatusField.setText((String)progress[1]);

break;

}

case 1:

{

if ((Boolean)progress[1])

{

MainActivity.NameField.setText("Connected with: " + NetworkAdapter.GetOtherName());

MainActivity.TransmitButton.setEnabled(true);

}

else

{

MainActivity.NameField.setText("Connected with: ---");

MainActivity.TransmitButton.setEnabled(false);

}

break;

}

case 2:

{

MainActivity.TransmitButton.setEnabled((Boolean)progress[1]);

MainActivity.ConnectButton.setEnabled((Boolean)progress[1]);

break;

}

case 3:

{

MainActivity.TransmitButton.performClick();

break;

}

}

}

@Override

protected void onPostExecute(Object result)

{

super.onPostExecute(result);

try

{

if (voice != null)

voice.stop();

}

catch (Exception e)

{

e.printStackTrace();

}

callback.onCallback(false);

}

}

**NetworkAdapter**

package com.vvv.wificall.wifi;

import java.lang.reflect.Method;

import com.vvv.wificall.event.CallbackEvent;

import com.vvv.wificall.event.ConditionalEvent;

import com.vvv.wificall.event.StopEvent;

import android.content.Context;

import android.net.ConnectivityManager;

import android.net.wifi.ScanResult;

import android.net.wifi.WifiConfiguration;

import android.net.wifi.WifiManager;

import android.preference.PreferenceManager;

import android.text.format.Formatter;

import android.util.Log;

public class NetworkAdapter

{

private static String OtherName = "---";

static String OtherClientIP = "0.0.0.0";

final static String NetworkSSID = "walkie";

final static String NetworkKey = "walkietalkie";

final static int StartWifiTimeout = 10000;

final static int StopWifiTimeout = 10000;

final static int StartApTimeout = 10000;

final static int StopApTimeout = 10000;

final static int BeginScanTimeout = 5000;

final static int BeginConnectionTimeout = 30000;

final static int WaitTimeSpan = 250;

static int ActionCounter = 0;

private static boolean ConditionalWait(int timeout, CallbackEvent statusCallback, StopEvent stopNotifier, ConditionalEvent event)

{

int ActionId = ActionCounter ++;

try

{

Log.d("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ precondition ]");

if (event.checkCondition())

return true;

Log.d("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ start ]");

if (!event.startEvent())

return false;

statusCallback.onCallback(event.getOperationName());

boolean status = false;

timeout /= WaitTimeSpan;

Log.d("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + "[ loop ]");

Log.v("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + "[ timeout " + (timeout \* WaitTimeSpan) + " ]");

while ((stopNotifier == null || !stopNotifier.isStopped()) && !(status = event.checkCondition()) && (timeout-- > 0))

{

try

{

Thread.sleep(WaitTimeSpan);

}

catch (Exception e)

{

e.printStackTrace();

}

Log.v("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + "[ timeout " + (timeout \* WaitTimeSpan) + " ]");

}

if (stopNotifier != null && stopNotifier.isStopped())

{

Log.i("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ cancelled ]");

event.onTimeout();

return false;

}

if (status)

{

Log.i("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ success ]");

return true;

}

else

{

Log.w("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ timeout ]");

event.onTimeout();

return false;

}

}

catch (Exception e)

{

Log.e("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ exception ]");

try

{

event.onTimeout();

}

catch (Exception e2)

{

e2.printStackTrace();

}

e.printStackTrace();

return false;

}

}

public static boolean StartWifi(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalWait(StartWifiTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

if (!StopAccessPoint(contextHandler, statusCallbackHandler, stopNotifierHandler))

return false;

return wifiManager.setWifiEnabled(true);

}

public boolean checkCondition() throws Exception

{

return (wifiManager.getWifiState() == WifiManager.WIFI\_STATE\_ENABLED);

}

public void onTimeout() throws Exception

{

StopWifi(contextHandler, statusCallbackHandler, null);

}

public String getOperationName()

{

return "Turning WiFi on...";

}

});

}

public static boolean StopWifi(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

return ConditionalWait(StopWifiTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

return wifiManager.setWifiEnabled(false);

}

public boolean checkCondition() throws Exception

{

return (wifiManager.getWifiState() == WifiManager.WIFI\_STATE\_DISABLED);

}

public void onTimeout() throws Exception

{

}

public String getOperationName()

{

return "Turning WiFi off...";

}

});

}

public static boolean StartAccessPoint(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalWait(StartApTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

if (!StopWifi(contextHandler, statusCallbackHandler, stopNotifierHandler))

return false;

Method SetWifiApEnabled = wifiManager.getClass().getMethod("setWifiApEnabled", WifiConfiguration.class, boolean.class);

WifiConfiguration netConfig = new WifiConfiguration();

netConfig.SSID = NetworkSSID;

netConfig.preSharedKey = NetworkKey;

netConfig.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.SHARED);

netConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);

netConfig.allowedProtocols.set(WifiConfiguration.Protocol.WPA);

netConfig.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA\_PSK);

netConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);

netConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);

netConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);

netConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);

return (Boolean)SetWifiApEnabled.invoke(wifiManager, netConfig, true);

}

public boolean checkCondition() throws Exception

{

Method IsWifiApEnabled = wifiManager.getClass().getMethod("isWifiApEnabled");

return ((Boolean)IsWifiApEnabled.invoke(wifiManager));

}

public void onTimeout() throws Exception

{

StopAccessPoint(contextHandler, statusCallbackHandler, null);

}

public String getOperationName()

{

return "Turning AP on...";

}

});

}

public static boolean StopAccessPoint(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

return ConditionalWait(StopApTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

Method SetWifiApEnabled = wifiManager.getClass().getMethod("setWifiApEnabled", WifiConfiguration.class, boolean.class);

return (Boolean)SetWifiApEnabled.invoke(wifiManager, null, false);

}

public boolean checkCondition() throws Exception

{

Method IsWifiApEnabled = wifiManager.getClass().getMethod("isWifiApEnabled");

return (!(Boolean)IsWifiApEnabled.invoke(wifiManager));

}

public void onTimeout() throws Exception

{

}

public String getOperationName()

{

return "Turning AP off...";

}

});

}

public static boolean BeginNetworkScan(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalWait(BeginScanTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

if (!StartWifi(contextHandler, statusCallbackHandler, stopNotifierHandler))

return false;

return wifiManager.startScan();

}

public boolean checkCondition() throws Exception

{

if (wifiManager.getScanResults() != null)

{

for (ScanResult result : wifiManager.getScanResults())

{

if (result.SSID.equals(NetworkSSID) || result.SSID.equals('"' + NetworkSSID + '"'))

return true;

}

}

return false;

}

public void onTimeout() throws Exception

{

StopWifi(contextHandler, statusCallbackHandler, null);

}

public String getOperationName()

{

return "Scanning for network...";

}

});

}

public static boolean BeginNetworkConnection(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final ConnectivityManager connectivityManager = (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalWait(BeginConnectionTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

if (!BeginNetworkScan(contextHandler, statusCallbackHandler, stopNotifierHandler))

return false;

if (wifiManager.getConfiguredNetworks() != null)

{

for (WifiConfiguration Network : wifiManager.getConfiguredNetworks())

{

if (Network.SSID.equals(NetworkSSID) || Network.SSID.equals('"' + NetworkSSID + '"'))

{

wifiManager.removeNetwork(Network.networkId);

break;

}

}

}

WifiConfiguration netConfig = new WifiConfiguration();

netConfig.SSID = '"' + NetworkSSID + '"';

netConfig.preSharedKey = '"' + NetworkKey + '"';

int networkId = wifiManager.addNetwork(netConfig);

if (networkId == -1)

return false;

return wifiManager.enableNetwork(networkId, true);

}

public boolean checkCondition() throws Exception

{

if (wifiManager.getConnectionInfo() == null)

return false;

if (wifiManager.getConnectionInfo().getSSID() == null)

return false;

if (!wifiManager.getConnectionInfo().getSSID().equals(NetworkSSID) &&

!wifiManager.getConnectionInfo().getSSID().equals('"' + NetworkSSID + '"'))

return false;

if (connectivityManager.getActiveNetworkInfo() == null)

return false;

if (!connectivityManager.getActiveNetworkInfo().isConnected())

return false;

return true;

}

public void onTimeout() throws Exception

{

StopWifi(contextHandler, statusCallbackHandler, null);

}

public String getOperationName()

{

return "Connecting to network...";

}

});

}

public static String GetOtherClientIP()

{

return OtherClientIP;

}

public static void SetOtherClientIP(String address)

{

OtherClientIP = address;

}

public static String GetUserName(Context context)

{

if (PreferenceManager.getDefaultSharedPreferences(context) == null)

return "User";

return PreferenceManager.getDefaultSharedPreferences(context).getString("nickNamePref", "User");

}

public static String GetOtherName()

{

return OtherName;

}

public static void SetOtherName(String name)

{

OtherName = name;

}

public static String GetServerIP(Context context)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

if (wifiManager.getDhcpInfo() == null)

return "0.0.0.0";

return Formatter.formatIpAddress(wifiManager.getDhcpInfo().gateway);

}

}

**SessionAdapter**

package com.vvv.wificall.wifi;

import android.util.Log;

public class SessionAdapter

{

public enum SessionMessage

{

MSG\_UNKNOWN(0),

MSG\_CONNECTION\_REQUEST(1),

MSG\_CONNECTION\_SUCCESS(2),

MSG\_CONNECTION\_FAILURE(3),

MSG\_TRANSMISSION\_BEGIN(4),

MSG\_TRANSMISSION\_PAYLOAD(5),

MSG\_TRANSMISSION\_END(6),

MSG\_DISCONNECT(7);

private int code = -1;

private SessionMessage(int c)

{

code = c;

}

public int getCode()

{

return code;

}

public String getDescription()

{

switch (SessionMessage.class.getEnumConstants()[code])

{

case MSG\_CONNECTION\_REQUEST: return "connection request";

case MSG\_CONNECTION\_SUCCESS: return "connected to server";

case MSG\_CONNECTION\_FAILURE: return "connection rejected";

case MSG\_TRANSMISSION\_BEGIN: return "transmission start";

case MSG\_TRANSMISSION\_PAYLOAD: return "transmission payload";

case MSG\_TRANSMISSION\_END: return "transmission end";

case MSG\_DISCONNECT: return "disconnect";

default: return "unknown message";

}

}

}

static int ActionCounter = 0;

public static SessionMessage PacketDispatcher(byte[] packet, byte[] payload)

{

int ActionId = ActionCounter ++;

String operationName = "Dispatching message (" + SessionMessage.MSG\_UNKNOWN.getDescription() + ")";

try

{

operationName = "Dispatching message (" + SessionMessage.class.getEnumConstants()[packet[0]].getDescription() + ")";

}

catch (Exception e)

{

e.printStackTrace();

}

Log.d("Session action", "Action # " + ActionId + ": " + operationName + " [ validating ]");

if (packet[1] != '#')

return SessionMessage.MSG\_UNKNOWN;

try

{

for (int i = 0; i < packet.length - 2; i ++)

payload[i] = packet[i + 2];

}

catch (Exception e)

{

e.printStackTrace();

}

Log.d("Session action", "Action # " + ActionId + ": " + operationName + " [ casting ]");

try

{

return SessionMessage.class.getEnumConstants()[packet[0]];

}

catch (Exception e)

{

e.printStackTrace();

}

return SessionMessage.MSG\_UNKNOWN;

}

public static byte[] PacketGenerator(SessionMessage message, byte[] payload)

{

int ActionId = ActionCounter ++;

byte[] packet = new byte[TransmissionAdapter.MaxPacketSize];

String operationName = "Generating message (" + message.getDescription() + ")";

Log.d("Session action", "Action # " + ActionId + ": " + operationName + " [ generating ]");

packet[0] = (byte)message.getCode();

packet[1] = (byte)'#';

if (payload == null)

return packet;

Log.d("Session action", "Action # " + ActionId + ": " + operationName + " [ payloading ]");

try

{

for (int i = 0; i < payload.length; i ++)

packet[i + 2] = payload[i];

}

catch (Exception e)

{

e.printStackTrace();

}

return packet;

}

}

**Transmission Adapter**

package com.vvv.wificall.wifi;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.ServerSocket;

import java.net.Socket;

import com.vvv.wificall.event.NotifierEvent;

import com.vvv.wificall.event.ObjectHolder;

import com.vvv.wificall.event.StopEvent;

import com.vvv.wificall.event.ThreadingEvent;

import android.media.AudioFormat;

import android.media.AudioRecord;

import android.util.Log;

public class TransmissionAdapter

{

public final static int MaxPacketSize = 2 + AudioRecord.getMinBufferSize(8000, AudioFormat.CHANNEL\_CONFIGURATION\_MONO, AudioFormat.ENCODING\_PCM\_16BIT);

final static int SendPacketTimeout = 5000;

final static int ReceivePacketTimeout = 5000;

final static int WaitTimeSpan = 250;

static int ActionCounter = 0;

static ServerSocket server = null;

static Socket socket = null;

private static boolean ConditionalThreading(int timeout, int timeSpan, StopEvent stopNotifier, ThreadingEvent event)

{

int ActionId = ActionCounter ++;

if (timeSpan < 0)

timeSpan = WaitTimeSpan;

final ThreadingEvent eventHandler = event;

final NotifierEvent threadNotifier = new NotifierEvent();

final Thread receiverThread = new Thread(new Runnable()

{

public void run()

{

try

{

eventHandler.startThread();

threadNotifier.setResultSuccess();

}

catch (Exception e)

{

threadNotifier.setResultFailure(e);

}

}

});

try

{

Log.d("Network action", "Action # " + ActionId + ": " + event.getOperationName() + " [ start ]");

receiverThread.start();

boolean status = false;

timeout /= timeSpan;

Log.d("Network action", "Action # " + ActionId + ": " + event.getOperationName() + "[ loop ]");

Log.v("Network action", "Action # " + ActionId + ": " + event.getOperationName() + "[ timeout " + (timeout \* timeSpan) + " ]");

while ((stopNotifier == null || !stopNotifier.isStopped()) && !(status = !receiverThread.isAlive()) && (timeout-- > 0))

{

try

{

Thread.sleep(timeSpan);

}

catch (Exception e)

{

e.printStackTrace();

}

Log.v("Network action", "Action # " + ActionId + ": " + event.getOperationName() + "[ timeout " + (timeout \* timeSpan) + " ]");

}

if (stopNotifier != null && stopNotifier.isStopped())

{

Log.i("Network action", "Action # " + ActionId + ": " + event.getOperationName() + " [ cancelled ]");

receiverThread.interrupt();

CloseConnections();

return false;

}

if (threadNotifier.getResult() == NotifierEvent.ThreadResult.Failure)

throw threadNotifier.getException();

if (status)

{

Log.i("Network action", "Action # " + ActionId + ": " + event.getOperationName() + " [ success ]");

return true;

}

else

{

Log.w("Network action", "Action # " + ActionId + ": " + event.getOperationName() + " [ timeout ]");

receiverThread.interrupt();

CloseConnections();

return false;

}

}

catch (Exception e)

{

Log.e("Network action", "Action # " + ActionId + ": " + event.getOperationName() + " [ exception ]");

receiverThread.interrupt();

CloseConnections();

e.printStackTrace();

return false;

}

}

private static void CloseConnections()

{

try

{

if (socket != null)

socket.close();

}

catch (Exception e)

{

e.printStackTrace();

}

try

{

if (server != null)

server.close();

}

catch (Exception e)

{

e.printStackTrace();

}

}

public static boolean SendPackets(byte[] packet, StopEvent stopNotifier, int timeout, int timeSpan)

{

final byte[] packetHandler = packet;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalThreading(((timeout >= 0) ? timeout : SendPacketTimeout), timeSpan, stopNotifier, new ThreadingEvent()

{

public void startThread() throws Exception

{

socket = null;

try

{

while (true)

{

try

{

socket = new Socket(NetworkAdapter.GetOtherClientIP(), 2012);

break;

}

catch (Exception e)

{

// Temporary workaround

//e.printStackTrace();

}

}

if (stopNotifierHandler != null && stopNotifierHandler.isStopped())

return;

OutputStream outputStream = socket.getOutputStream();

outputStream.write(packetHandler);

}

catch (Exception e)

{

throw e;

}

finally

{

CloseConnections();

}

}

public String getOperationName()

{

return "Sending data...";

}

});

}

public static boolean ReceivePackets(byte[] packet, ObjectHolder<String> senderIPHolder, StopEvent stopNotifier, int timeout, int timeSpan)

{

final ObjectHolder<String> senderIPHolderHandler = senderIPHolder;

final ObjectHolder<byte[]> packetHolder = new ObjectHolder<byte[]>();

final StopEvent stopNotifierHandler = stopNotifier;

boolean result = ConditionalThreading(((timeout >= 0) ? timeout : ReceivePacketTimeout), timeSpan, stopNotifier, new ThreadingEvent()

{

public void startThread() throws Exception

{

server = null;

socket = null;

try

{

while (true)

{

try

{

server = new ServerSocket(2012);

break;

}

catch (Exception e)

{

// Temporary workaround

//e.printStackTrace();

}

}

if (stopNotifierHandler != null && stopNotifierHandler.isStopped())

return;

socket = server.accept();

InputStream inputStream = socket.getInputStream();

byte[] inputBuffer = new byte[MaxPacketSize];

inputStream.read(inputBuffer);

if (senderIPHolderHandler != null)

senderIPHolderHandler.setObject(socket.getInetAddress().getHostAddress());

packetHolder.setObject(inputBuffer);

}

catch (Exception e)

{

throw e;

}

finally

{

CloseConnections();

}

}

public String getOperationName()

{

return "Receiving data...";

}

});

if (result)

{

try

{

for (int i = 0; i < packet.length; i ++)

packet[i] = packetHolder.getObject()[i];

}

catch (Exception e)

{

e.printStackTrace();

}

}

return result;

}

}

**Callback event**

package com.vvv.wificall.event;

public interface CallbackEvent

{

abstract void onCallback(Object arg);

}

**Conditional Event**

package com.vvv.wificall.event;

public interface ConditionalEvent

{

abstract boolean startEvent() throws Exception;

abstract boolean checkCondition() throws Exception;

abstract void onTimeout() throws Exception;

abstract String getOperationName();

}

**Notifier Event**

package com.vvv.wificall.event;

public class NotifierEvent

{

public enum ThreadResult { Pending, Success, Failure }

private ThreadResult result = null;

private Exception exception = null;

public NotifierEvent()

{

result = ThreadResult.Pending;

}

public void setResultSuccess()

{

if (result == ThreadResult.Pending)

result = ThreadResult.Success;

}

public void setResultFailure(Exception e)

{

if (result == ThreadResult.Pending)

{

result = ThreadResult.Failure;

exception = e;

}

}

public ThreadResult getResult()

{

return result;

}

public Exception getException()

{

return exception;

}

}

**Object Holder**

package com.vvv.wificall.event;

public class ObjectHolder<Type>

{

private Type value = null;

public void setObject(Type newObject)

{

value = newObject;

}

public Type getObject()

{

return value;

}

}

**StopEvent**

package com.vvv.wificall.event;

public class StopEvent

{

private boolean stop = false;

public StopEvent()

{

stop = false;

}

public void stop()

{

stop = true;

}

public void reset()

{

stop = false;

}

public boolean isStopped()

{

return stop;

}

}

**ThreadingEvent**

package com.vvv.wificall.event;

public interface ThreadingEvent

{

abstract void startThread() throws Exception;

abstract String getOperationName();

}