



Developer's guide

Philips SpeechAir App Software Development Kit 1.0.005

PHILIPS

This page is intentionally left blank

- 1. Introduction 3**
- 2. SpeechAir classes..... 3**
- 3. Test App 3**
- 4. SpeechAir methods..... 5**
 - 4.1. USB Connection 5
 - 4.1.1. Use *SpeechAir_prohibitMSC()* to prohibit USB connection..... 5
 - 4.1.2. Use *SpeechAir_allowMSC()* to allow USB connection again 6
 - 4.2. Read the slider position and smart button..... 7
 - 4.3. Select and activate microphones 10
 - 4.4. Set microphone gain..... 11
 - 4.5. Select and activate back speaker 11
 - 4.6. LED control 12
 - 4.7. Call PDR from another app 13
- 5. USB driver installation for Android studio 14**
 - 5.1. USB driver installation 14
 - 5.2. Importing jar libraries into Android studio 16

1. Introduction

Philips SpeechAir is a smart voice recorder running on an Android platform. The SpeechAir App SDK is an API which provides a range of interfaces. It can be used to integrate communication directly between a single SpeechAir device and the device application. With the SpeechAir App SDK, you have full access to all SpeechAir-specific hardware elements (LEDs, Slide switch, Microphones, and many more). Philips SpeechAir uses the Android SDK level 19.

2. SpeechAir classes

Please import following classes in your project:

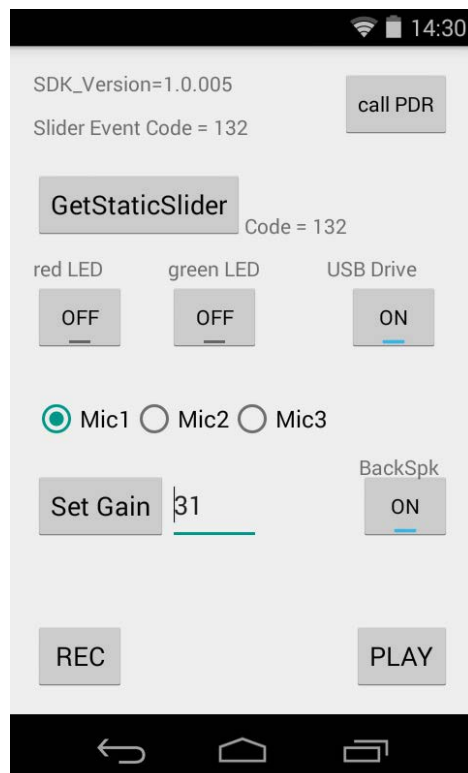
```
import com.sps.setmicgain.SetMicGain;
import com.sps.slidekey.SlideKey;
import com.sps.switchmic.SwitchMic;
import com.sps.switchled.SwitchLed;
```

Source code is delivered with the device and saved on the USB Stick in folder \App SDK and Source code - V1.0.005\SpeechAir_SDK 1.0.005

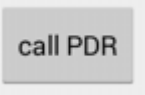
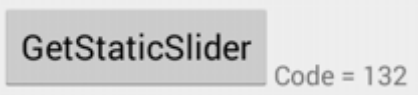



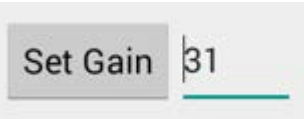


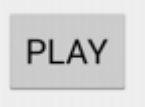
3. Test App

We have developed a test app that includes all functionalities of the SDK. Find the source code of the test app on the USB Stick delivered with SpeechAir.

Install and start the test app on the SpeechAir – The UI looks like this:

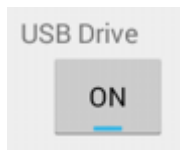


Following functionalities have been built into the test app.

| UI element | Functionality | Chapter |
|---|--|---|
|  | Use the Philips Dictation Recorder App | Call PDR from another app |
|  | GetStatic Slider returns current status of the slider | Read the slider position and smart button |
|  | Turn on & off LED | LED control |
|  | Allow/Prohibit USB connection. | USB Connection |
|  | Switch between mics | Select and activate microphones |
|  | Set microphone sensitivity | Set microphone gain |
|  | Toggle between front and back speaker | Select and activate back speaker |
|  | Record a file in .wav format (PCM, 16kHz, 16bit) | - |
|  | Playback of recorded file | - |

4. SpeechAir methods

4.1. USB Connection



The lines below demonstrate how to prevent SpeechAir from connecting to the USB host. This is useful if you don't want your recording to be interrupted by the USB connection e.g.: call the method `SpeechAir_prohibitMSC()` before you start a recording, and `SpeechAir_allowMSC()` when you stop it again.

This is necessary because the file-system gets unmounted from the Android-OS while connected to an USB-host, in order to serve as a mass storage drive for the USB-host.

It is important to save the return value from `SpeechAir_prohibitMSC()` in order to pass this value as parameter when you call `SpeechAir_allowMSC()`.

This way the Operating System can track if all the `SpeechAir_allowMSC()`-calls have been made for every single call of `SpeechAir_prohibitMSC()`. This ensures that USB connection is only allowed if every single caller of `SpeechAir_prohibitMSC()` has allowed the connection again. That is important if different threads are using this methods to prohibit USB connection.

The `SpeechAir_prohibitMSC()` method must pass the `PackageName` as parameter. This way the OperatingSystem can guarantee that it will not wait any longer for the corresponding `SpeechAir_allowMSC()`, if the app crashes for some reason.

Please make sure to prevent USB connection every time you access the file-system via your app. i.e.: read or write a file via your app.

4.1.1. Use `SpeechAir_prohibitMSC()` to prohibit USB connection

```
private static final String SA_UMS_SERVICE_DES = "saumsservice";
private static long prohibitID;
```

```
prohibitID = ((SaUmsManager)
getSystemService(SA_UMS_SERVICE_DES)).SpeechAir_prohibitMSC(this.getPackageName());
```

| | |
|--------------|---|
| Parameter | make sure to use the <code>PackageName</code> |
| Return value | a unique long value (<code>prohibitID</code> in the above example) |

4.1.2. Use *SpeechAir_allowMSC()* to allow USB connection again

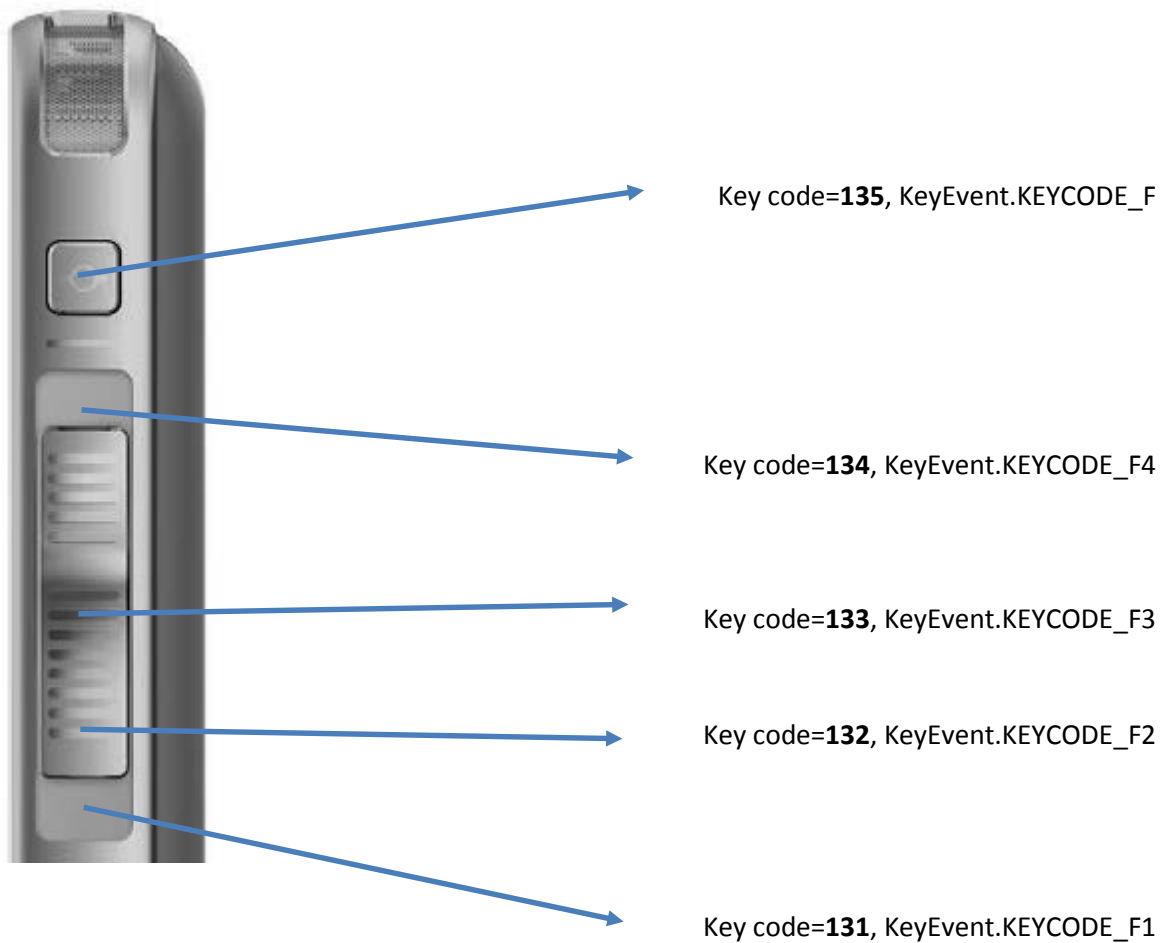
```
((SaUmsManager)getService(SA_UMS_SERVICE_DES)).SpeechAir_allowMSC(prohibitID);
```

| | |
|--------------|--|
| Parameter | make sure to use the previously stored return value of the <i>SpeechAir_prohibitMSC()</i> method (<i>prohibitID</i> in the above example) |
| Return value | none |

4.2. Read the slider position and smart button

```
GetStaticSlider Code = 132
```

The lines below demonstrate how to read out the slider and hardware-button events. Please keep in mind that Android-events are only fired if the state of the key changes. You can not use this method to detect the initial state of the slider when your app starts.



Override the `onKeyDown()` method to receive the above described keyCodes and KeyEvents:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    sliderStateTextView.setText(keyCode+"");

    switch(keyCode)
    {
```



```

        case KeyEvent.KEYCODE_F5: //Smart
Button (key above the slider)
        Log.i("SpeechAir_SDK", "smart button keycode=" +
keyCode);
        return(true);

        case KeyEvent.KEYCODE_F4: //Slider at
top
        Log.i("SpeechAir_SDK", "slider top keycode=" +
keyCode);
        return(true);

        case KeyEvent.KEYCODE_F3: //Slider
below top
        Log.i("SpeechAir_SDK", "slider below top keycode=" +
keyCode);
        return(true);

        case KeyEvent.KEYCODE_F2: //Slider
above bottom
        Log.i("SpeechAir_SDK", "slider above bottom keycode=" +
keyCode);
        return(true);

        case KeyEvent.KEYCODE_F1: //Slider at
bottom
        Log.i("SpeechAir_SDK", "slider bottom keycode=" +
keyCode);
        return(true);

        case KeyEvent.KEYCODE_VOLUME_UP:
        Log.i("SpeechAir_SDK", "volume up keycode=" + keyCode);
        return super.onKeyDown(keyCode, event); //
KEYCODE_VOLUME_UP is available for all other apps
        //return(true); //
KEYCODE_VOLUME_UP is not forwarded to other apps

        case KeyEvent.KEYCODE_VOLUME_DOWN:
        Log.i("SpeechAir_SDK", "volume down keycode=" +
keyCode);
        return super.onKeyDown(keyCode, event); //
KEYCODE_VOLUME_UP is available for all other apps
        //return(true); //
KEYCODE_VOLUME_UP is not forwarded to other apps

        default:
            break;
    }

    return(super.onKeyDown(keyCode, event));
}

```

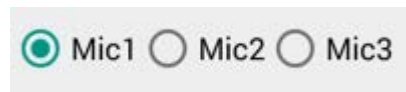
Use the method `getSliderState()` in order to read out the current state of the slide switch. eg: this method can be used to read out the slider state when the app starts. This method returns the `keyCode` as int value. Make sure to import the `com.sps.slidekey.SlideKey` class.

```
import com.sps.slidekey.SlideKey;
```

```
keyCode = SlideKey.getSliderState();
```

| | |
|--------------|----------------------|
| Parameter | none |
| Return value | keyCode as int value |

4.3. Select and activate microphones



Use the method `SwitchMic.switchMicTo(context, value)` to choose the desired microphone. Make sure to import the `com.sps.switchmic.SwitchMic` class.

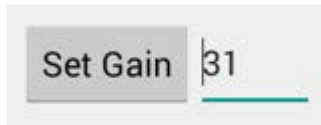
```
import com.sps.switchmic.SwitchMic;

mcontext = getApplicationContext();
SwitchMic.switchMicTo(mcontext, 1);
```

- Parameter1 the application context
- Parameter2 microphone value
- Return value none

| Paramter 2 (microphone value) | Microphone |
|-------------------------------|--|
| 1 | Dictation microphone - unidirectional |
| 2 | Telephony microphone - omnidirectional |
| 3 | Dictation microphone - omnidirectional |

4.4. Set microphone gain



Use the method `SetMicGain.setGainValue()` to set the desired microphone gain. Make sure to import the `com.sps.setmicgain.SetMicGain` class.

```
import com.sps.setmicgain.SetMicGain;
```

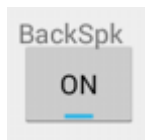
```
SetMicGain.setGainValue(MainActivity.this, gain_value);
```

Parameter1 the activity
 Parameter2 gain value
 Return valu... none

The microphone gain can be set to a value from 0 to 31.

| Paramter 2 (gain value) | Gain |
|-------------------------|-------------------------|
| 0 | Minimum microphone gain |
| | |
| 31 | Maximum microphone gain |

4.5. Select and activate back speaker



The lines below demonstrate how to turn the large back-speaker on/off.

```
audioManager =
(AudioManager)mcontext.getSystemService(Context.AUDIO_SERVICE);
audioManager.setMode(AudioManager.STREAM_MUSIC);

audioManager.setSpeakerphoneOn(true); //turn the back speaker on
audioManager.setSpeakerphoneOn(false); //turn the back speaker off
```

4.6. LED control



The lines below demonstrate how to control the red and green LED of SpeechAir above the telephone speaker.

Make sure to import the `com.sps.switchled.SwitchLed` class.

```
import com.sps.switchled.SwitchLed;
```

```
SwitchLed.turnOnRedLED();  
SwitchLed.turnOffRedLED();  
SwitchLed.turnOnGreenLED();  
SwitchLed.turnOffGreenLED();
```

4.7. Call PDR from another app



The lines below demonstrate how to start the Philips Dictation Recorder App (PDR) and hand over some meta-data. The PDR will start a new recording and store the meta-data in the XML file of the recording. After the user presses the EOL (End Of Letter) button in the PDR, the audio-file and XML-file are copied to this directory: `root\Android\data\com.speech\files\transfer\`

```
Intent launchIntent =
getPackageManager().getLaunchIntentForPackage("com.speech");
launchIntent.putExtra("com.speech.package", "com.myapp");
launchIntent.putExtra("com.speech.author", "myAuthor");
launchIntent.putExtra("com.speech.worktype", "myWorktype");
launchIntent.putExtra("com.speech.category", "myCategory");
launchIntent.putExtra("com.speech.barcode", "myBarcode");
launchIntent.putExtra("com.speech.attribute1", "myAttribute1");
launchIntent.putExtra("com.speech.attribute2", "myAttribute2");
launchIntent.putExtra("com.speech.attribute3", "myAttribute3");
launchIntent.putExtra("com.speech.attribute4", "myAttribute4");
launchIntent.putExtra("com.speech.attribute5", "myAttribute5");
launchIntent.putExtra("com.speech.comment", "myComment");
launchIntent.putExtra("com.speech.list_usage1", "myListUsage1");
launchIntent.putExtra("com.speech.list_column1", "myListColumn1");
launchIntent.putExtra("com.speech.list_usage2", "myListUsage2");
launchIntent.putExtra("com.speech.list_column2", "myListColumn2");
launchIntent.putExtra("com.speech.list_usage3", "myListUsage3");
launchIntent.putExtra("com.speech.list_column3", "myListColumn3");
launchIntent.putExtra("com.speech.list_usage4", "myListUsage4");
launchIntent.putExtra("com.speech.list_column4", "myListColumn4");
launchIntent.putExtra("com.speech.list_usage5", "myListUsage5");
launchIntent.putExtra("com.speech.list_column5", "myListColumn5");
launchIntent.putExtra("com.speech.list_usage6", "myListUsage6");
launchIntent.putExtra("com.speech.list_column6", "myListColumn6");
launchIntent.putExtra("com.speech.list_usage7", "myListUsage7");
launchIntent.putExtra("com.speech.list_column7", "myListColumn7");
launchIntent.putExtra("com.speech.list_usage8", "myListUsage8");
launchIntent.putExtra("com.speech.list_column8", "myListColumn8");
launchIntent.putExtra("com.speech.list_usage9", "myListUsage9");
launchIntent.putExtra("com.speech.list_column9", "myListColumn9");
launchIntent.putExtra("com.speech.list_usage10", "myListUsage10");
launchIntent.putExtra("com.speech.list_column10",
"myListColumn10");
```

If a `putExtra` with `"com.speech.package"` as name has been defined during the call of the `LaunchIntent` of the PDR, the value of the `putExtra` will be used as the package name which will be launched after the user has pressed EOL (in the above example the package with the name `"com.myapp"` would be launched).

This way the calling App could specify its own package name as value for the `"com.speech.package"`-`putExtra` and the PDR would start the calling App again with following `EXTRA_MESSAGE`:

```
name="com.speech.recording_finished"
```

value=string containing the full path+name of the recording in the transfer-folder

The calling app can use this information to further process this file.
eg.: upload to a server, send via e-mail, etc.

5. USB driver installation for Android studio

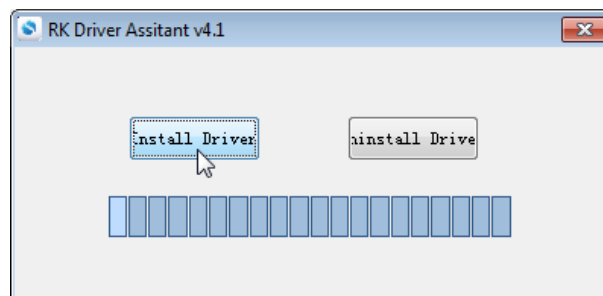
Before releasing an app, make sure to test the app on SpeechAir. Find a description on how to connect SpeechAir with the Android studio installed on a Windows computer.

5.1. USB driver installation

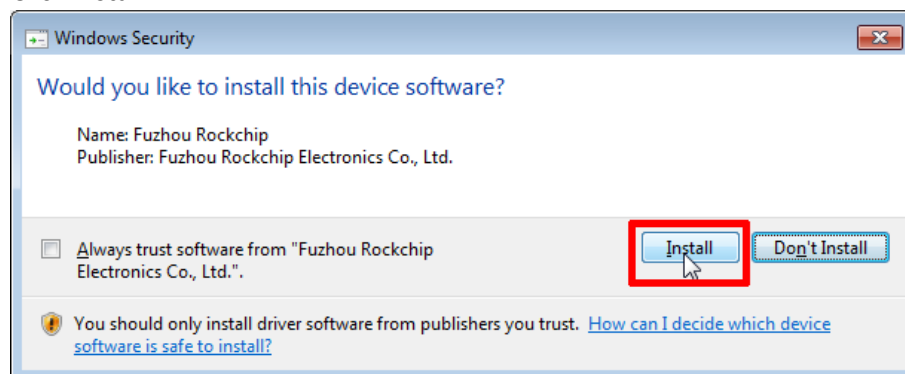
Start the installation by doubleclicking DriverInstall.exe in the "DriverAssistant_V4.1.1" folder

| Name | Date modified | Type | Size |
|-------------------|------------------|-----------------------|--------|
| ADBDriver | 01.06.2015 21:32 | File folder | |
| bin | 01.06.2015 21:32 | File folder | |
| Driver | 01.06.2015 21:32 | File folder | |
| config.ini | 26.09.2014 10:19 | Configuration sett... | 1 KB |
| DriverInstall.exe | 26.09.2014 10:19 | Application | 484 KB |
| Readme.txt | 26.09.2014 10:19 | Text Document | 1 KB |

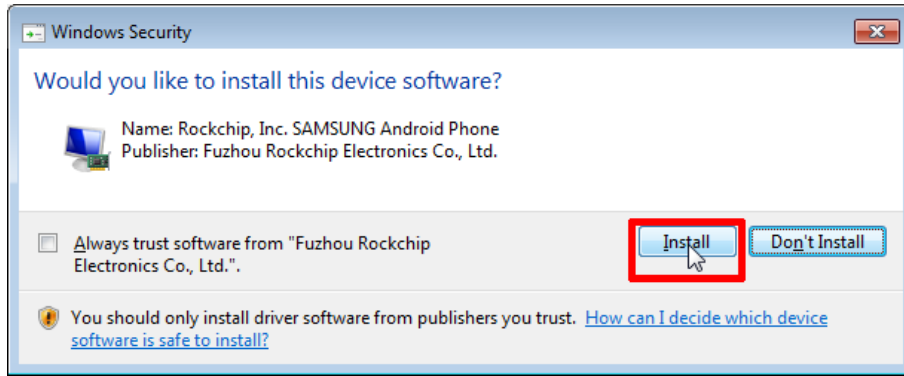
Installation will start



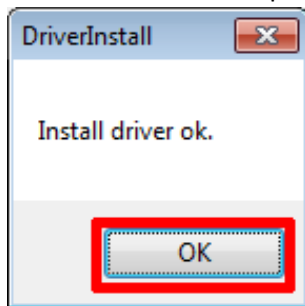
Click Install



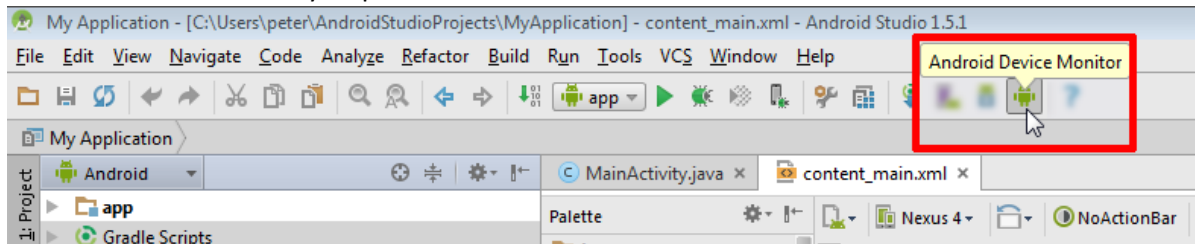
Click Install again



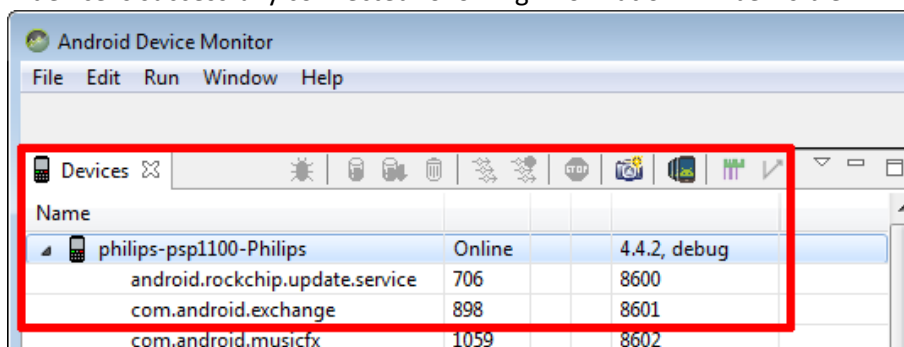
Finish the installation process clicking OK



Check if installed correctly: Open Andorid studio and Click Android Device Monitor

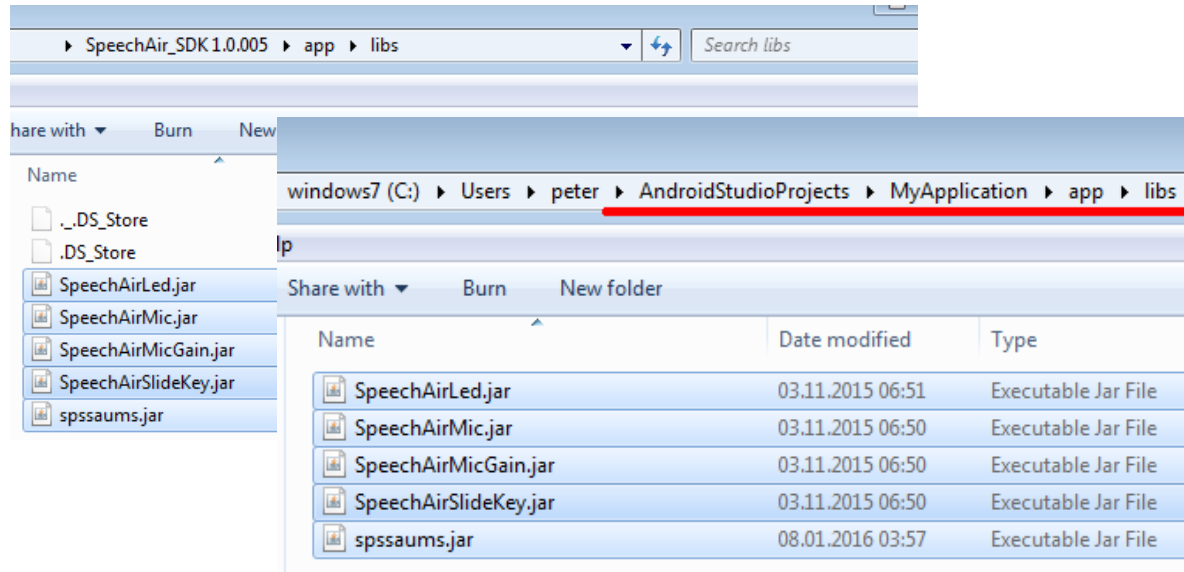


If device is successfully connected following information will be visible



5.2. Importing jar libraries into Android studio

Open the SDK libs-folder, select the files and copy them in your libs project folder



Specifications are subject to change without notice.

Last Changes – Speech Processing Solutions GmbH Support Group Vienna

Trademarks are the property of Royal Philips or their respective owners.

© 2016 Royal Philips. All rights reserved.

Document Version 1.0, 24.02.2016