Customizing Irdroid™

Secrets and tips for programming
Tricks handbook

This small handbook is an effort to describe in detail the Irdroid project – an open source, open hardware infrared remote control for the Google's Android operating system. In the first place I would like to thank my wife who encouraged me and supported me in writing this book. I would like to thank also Zokama (http://www.zokama.com) who actually ported the LIRC code for Android. Last but not least I would like to thank the reader and those people who purchased the Irdroid module, the Kit or this book. Thank you!

For the convenience of the reader, the links in this book are available also as a QR barcodes. The QR barcode can be scanned via the barcode scanner of your Android device. If you don't have a barcode scanner you could download one from the Google's Android Market. The barcodes in this book will save you typing in the links to your Android device.

INTRODUCTION 6

TOOLS AND SOFTWARE 7

LET'S START WITH THE HARDWARE: 10

OPERATION PRINCIPLE: 11
BUILDING THE IRDROID MODULE 14

INSTALLING ECLIPSE AND ADT PLUG-IN FOR ECLIPSE 17

PREPARING YOUR DEVELOPMENT COMPUTER 18
DOWNLOADING THE ADT PLUG-IN 21

DOWNLOADING AND COMPILING THE SOURCE. 23

CONFIGURATION OF THE ANDROID VIRTUAL DEVICE EMULATOR (AVD) 28

CHANGING THE DEFAULT AVD SKIN 30

DOWNLOADING NEW PLATFORM SKINS: 31
IRDROID AVD SKINS LIBRARY 35
INSTALLING AVD SKINS. 36

RUNNING IRDROID 38

TESTING IRDROID ON THE EMULATOR 40

CUSTOMIZING IRDROID SOURCE CODE 41

THE "LIRC.JAVA" CLASS SOURCE CODE 41
THE "ICONIC" ACTIVITY SOURCE CODE 42
THE IRDROID ACTIVITY SOURCE CODE 44

GLOSSARY 63

TECHNICAL SPECIFICATIONS – IRDROID V.1.0 64

IRDROID™ V.1.0 TECHNICAL DATA: 64
IRDROID™ V.1.0 MODULE APPS COMPATIBILITY: 64
APP SPECIFICATIONS: 65

REFERENCES: 66

***ABOUT THE AUTHOR* 67**

Introduction

Irdroid is an open source universal infrared remote control for Android. The project comprises a simple Android app and hardware module called Irdroid v1.0 which makes it possible to control various Infrared devices. The project uses code from the well known LIRC (Linux Infrared Remote Control) and from Zokama. The Irdroid application is fully compatible with LIRC conf files and can be downloaded from the official Irdroid website as well as from the Android Market.

This book will reveal the secrets of customizing the Irdroid source code as well as a detailed explanation of the tools used to design the Irdroid hardware and software. You will learn how to configure your development environment, how to use GIT and compile / run your custom Irdroid app on a fully functional Android Emulator for a PC / MAC. You will also discover the operation principle of the Irdroid v1.0 Hardware module.

Tools and Software

The Irdroid Development environment can be practically setup for minutes. In the Basic setup you will need to install the Android SDK, the latest version of JRE and possibly Eclipse (If you intend to use Eclipse as an IDE. Most of the Examples in this handbook are with Eclipse running on Windows XP, however Eclipse runs also on other operating systems.

If you intend to change the ported LIRC library for Irdroid then you will need to install also the Android NDK (Native Development Kit).

This book will not detail the installation of the Android NDK or the Eclipse as these are already available on the Internet. You may have a look at the following links on the Internet in order to learn how to configure your development environment properly. The Android Software development kit can be downloaded from:

http://developer.android.com/sdk/index.html ,

The link also contains step by step information on how to configure it. JRE – Java Runtime Environment can be downloaded from here:

http://www.java.com/en/download/index.jsp

Eclipse:

http://www.eclipse.org/helios/

Android NDK can be downloaded from this website:
http://developer.android.com/sdk/ndk/index.html

At the end you will also need to download and install the GIT repository plug-in for Eclipse from the Eclipse plug-in website.

After you install the above mentioned software kits you are almost ready to start develop and test your custom Irdroid app. If you don't have an Irdroid module yet, you may download the Irdroid module schematics / PCB files from the Irdroid website and make a DIY module.

You have also the option to purchase a kit or a built and tested Irdroid module, if available (check http://www.irdroid.com).

QR Code (alternate text – http://www.Irdroid.com)

Let’s start with the Hardware:

The module’s main task is to amplify the signal, generated from the app and to provide an IR interface to the relevant Android device. The active amplification is necessary, because the output signal from an Android device is not powerful enough to light up IR LEDs, as well as to provide a decent remote control range.

The module practically amplifies the generated waveform from the app and emits IR Light via the IR LEDs at 940nm wavelength. The input of the module is provided by the Android Device 3.5mm Audio jack.

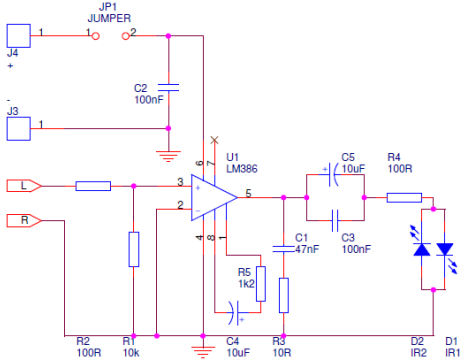
The Left and The Right audio channels are used, (GND) is not connected. The amplification is done using an inexpensive LM386-M1 mono audio amplifier which is configured for a gain of 200 times.

This configuration assures enough power @ 6V in order to achieve a remote control distance of about 10 meters.

Operation principle:

The Irdroid schematic is shown on **figure 1**. The Audio Signal, generated from the app is amplified via the LM-386-M1 Audio Amplifier and it is fed to the IR LEDs. Then the signal is emitted via the IR LEDs at 940nm.

Figure 1 - Irdroid 1.0 schematic

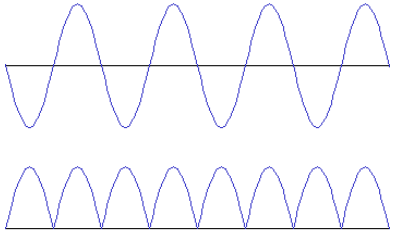


The Irdroid app is responsible for generating a 19kHz audio tone. The infared data is modulated

on the 19kHz sine wave. The signal is amplified via the LM386 audio amplifier and rectified via the two IR leds, doubling the frequency to 38kHz (Figure 2). The first IR led rectifies the positive halfwave of the audio signal and the second IR led the negative halfwave of the signal.

The LM386 mono audio amplifier is configured to amplify the signal 200 times so that the radiated IR light power is enought to achieve a remote control distance of about 10 meters.

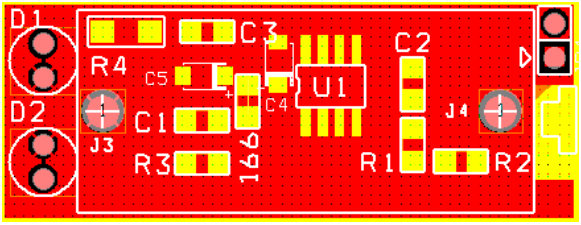
Figure 2. The Audio signal before and after rectification



The top wave shows how the signal comes from the Android device soundcard. The second is how

it looks like after amplification and rectification. As you can see, the singal frequency is doubled.

Figure 3. – Irdroid v 1.0 pcb



D1 and D2 from figure 3 are the two IR LEDs, U1 is the amplifier IR – LM386-M1, and the other components are capacitors and resistors detailed in the schematic.

Most of the components are SMD (surface mount) only the Jumper, the two IR Leds, the Battery Holder and the audio jack are coventional parts. The download section at http://www.irdroid.com contains a zip archive of the schematics and the production files of the module.

You could use the schematics and the production files to produce boards using your favourite printed

circuit board manufacturer. In most of the companies offering pcb manufacturing service you will find out that they could also solder the SMD components for you, as well as to build complete modules.
There are some online services for pcb manufacturing / production. Consider using Google to find out which is the best for you.

I personally use Vprint-pcb and Olimex for the pcb manufacturing as these companies offer low cost service and a great quality.

Some suggestions for PCB Manufacturing:

- Olimex LTD – http://www.olimex.com
- Custompcb – http://www.custompcb.com
- Futurlec – http://www.futurlec.com

Building the Irdroid module

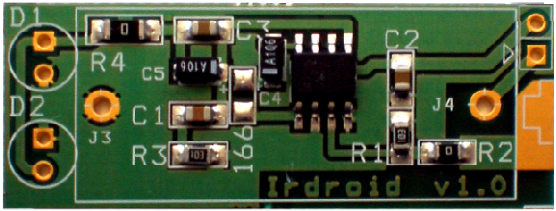
Building the Irdroid module will require at least rudimentary soldering skills, soldering iron and willingness to experiment. To produce Irdroid

board / boards you could use the production files available at the Irdroid website and make an enquiry about your favorite pcb manufacturing company. You could also check the Irdroid website for module availability. The modules available through the Irdroid website are as follows:

Irdroid Kit.

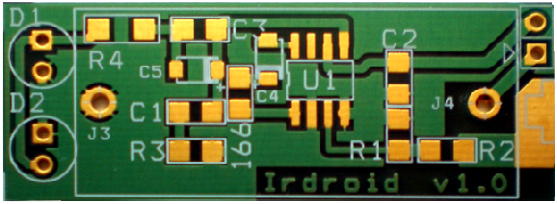
The Irdroid KIT is a development kit suitable for people who want to make the DIY module. The kit includes an Irdroid v1.0 PCB with soldered SMD components, a battery holder, a jumper, 2 IR LEDS and a 3.5mm Audio jack.

Irdroid PCB with soldered SMD components:



The Irdroid kit includes a pcb with soldered SMD components, 2 IR LEDS and a jumper. This kit is for those of you who intend to use another power supply (not the standard battery) or just want to experiment with the Irdroid.

Irdroid PCB



The Irdroid PCB without components (PCB only)

Installing Eclipse and ADT plug-in for Eclipse

If you will be developing in Eclipse with the Android Development Tools (ADT) Plug-in—the recommended path if you are new to Android—make sure that you have a suitable version of Eclipse installed on your computer.

If you need to install Eclipse, you can download it from this location:

http://www.eclipse.org/downloads/

The "Eclipse Classic" version is recommended. Otherwise, a Java or RCP version of Eclipse is recommended.

Android Development Tools (ADT) is a plug-in for the Eclipse IDE that is designed to give you a powerful, integrated environment in which to build Android applications.

ADT extends the capabilities of Eclipse to let you quickly set up new Android projects, create an

application UI, add components based on the Android Framework API, debug your applications using the Android SDK tools and even export signed (or unsigned) .apk files in order to distribute your application.

Developing in Eclipse with ADT is highly recommended and is the fastest way to get started. With the guided project setup it provides, as well as tools integration, custom XML editors, and debug output pane, ADT gives you an incredible boost in developing Android applications.

Preparing Your Development Computer

The SDK starter package is not a full development environment—it includes only the core SDK Tools, which you can use to download the rest of the SDK components (such as the latest Android platform).
If you do not have it already, get the latest version of the SDK starter package. Get it from:

http://developer.Android.com/sdk/index.html .

If you downloaded a .zip or .tgz package (instead of the SDK installer), unpack it to a safe location on

your machine. By default, the SDK files are unpacked into a directory named Android-sdk-<machine-platform>.
If you downloaded the Windows installer (.exe file), run it now and it will check whether the proper Java SE Development Kit (JDK) is installed (installing it, if necessary), then install the SDK Tools into a default location (which you can modify).

Make a note of the name and location of the SDK directory on your system—you will need to refer to the SDK directory later, when setting up the ADT plug-in and when using the SDK tools from the command line.

ADT is a plug-in for the Eclipse IDE. Before you can install or use ADT, you must have a compatible version of Eclipse installed on your development computer. If Eclipse is already installed on your computer, make sure that it is a version that is compatible with ADT and the Android SDK.

- If you need to install or update Eclipse, you can download it from this location:

http://www.eclipse.org/downloads/

The "Eclipse Classic" version is recommended. Otherwise, a Java or RCP version of Eclipse is recommended.

Additionally, before you can configure or use ADT, you must install the Android SDK starter package, as described in:

http://developer.Android.com/sdk/installing.html#Installing

Specifically, you need to install a compatible version of the Android SDK Tools and at least one development platform. To simplify ADT setup, we recommend installing the Android SDK prior to installing ADT.

When your Eclipse and Android SDK environments are ready, continue with the ADT installation as described in the steps below.

Downloading the ADT Plug-in

Use the Update Manager feature of your Eclipse installation to install the latest revision of ADT on your development computer.

1. Start Eclipse, then select **Help** > **Install New Software...**.
2. Click **Add**, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*:

   https://dl-ssl.google.com/Android/eclipse/

4. Click **OK**

Note: If you have trouble acquiring the plug-in, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).

5. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
6. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.

7. Read and accept the license agreements, then click **Finish**.

Note: If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.

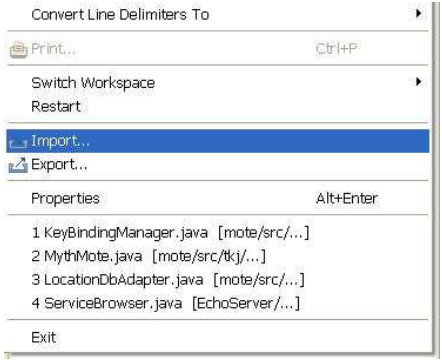
8. When the installation is complete, restart Eclipse.

Downloading and compiling the source.

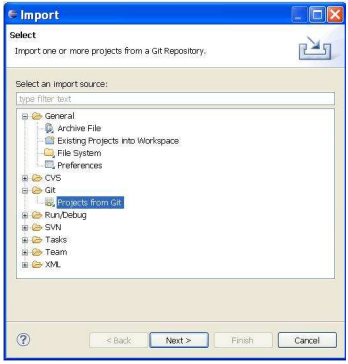
The Irdroid source code is available for download at http://www.github.com/Irdroid

Here you can find out how to import the source code in your development environment using Eclipse and the GIT plug-in for Eclipse.

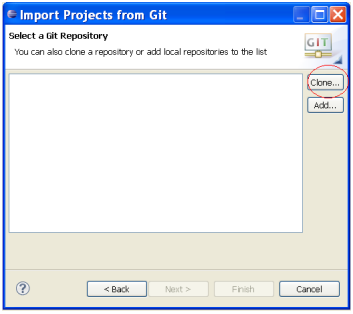
Start Eclipse and select file - > import



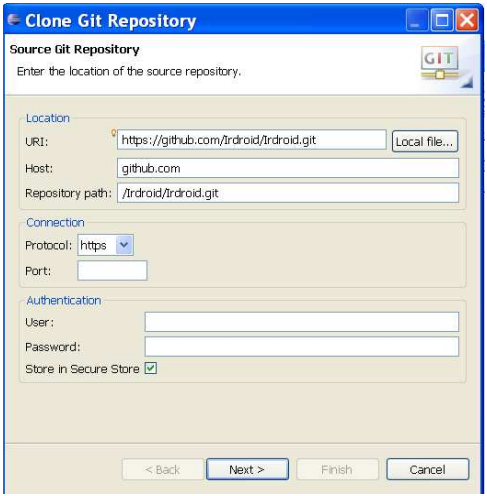
Afterwards select the “Import projects from GIT”



Click “Clone”



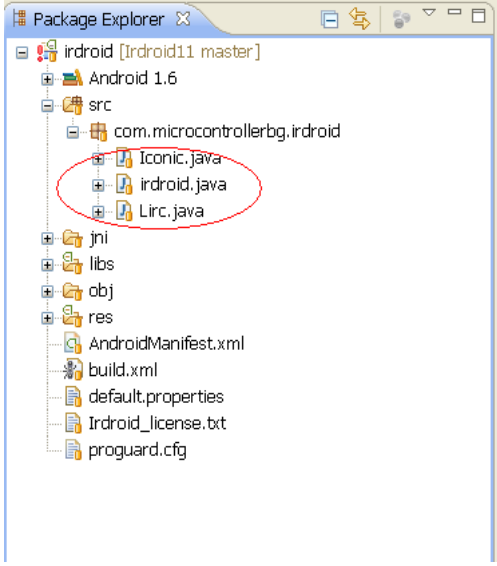
Enter the Following “URI” and click next.



After you click next and select the default options, the project import process will start and the Irdroid source code will be imported in your development environment.

At this point if your development environment is configured properly, you should be able to compile the source.

The Irdroid project consists of two Android activities, one class and 1 native Android library.



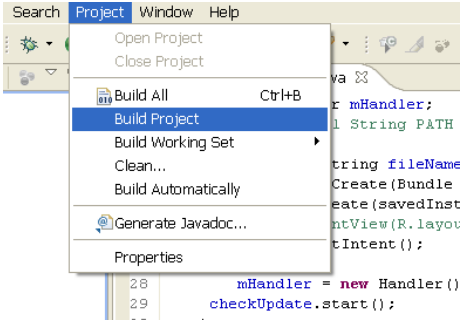
Activities:

- The main activity – Irdroid
- The Update Database Activity – Iconic

Classes:

- The Lirc.java calls to libIrdroid.so (the port of LIRC)

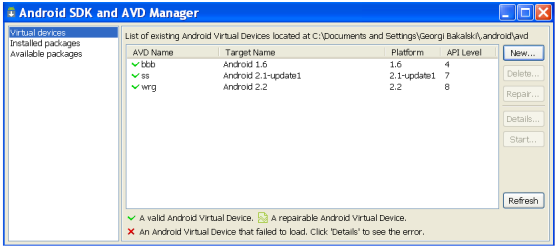
If your development environment is configured correctly you can build the project by clicking Project - > Build



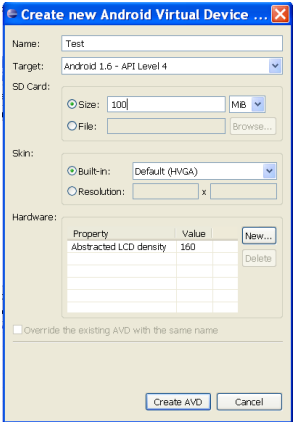
After a successful compilation you should be able to run the project on the emulator that comes with the Android Software Development Kit. But first you need to define the new AVD (Android Virtual Device).

Configuration of the Android virtual device emulator (AVD)

To play with the emulator, first you need to do some configurations according to your particular requirements. This includes the target Android platform version that you want to emulate, the size of the emulated storage (the SDCARD), the size of the screen etc. To start the AVD configuration manager, click “window-> Android SDK and AVD manager”.



Configure a new AVD according to your needs by clicking “New”. You will be able to select the new AVD parameters like name, storage size, Android version and screen details.



The “Create new Android Virtual Device” dialog provides you with the options to select the platform version to be emulated, the storage size, as well as the skin that you want to use.

The next chapter deals separately with downloading and installing skins for the Android emulator AVD.

Changing the default AVD Skin

Any Android developer knows that the default Google's Android emulator comes with a set of loveless default looks.

Fortunately there are skins available on the Internet which and that changes the whole picture. You could download a set of fabulous Nexus skins for the Android emulator from:

These skins are really fabulous with a cool glow effect and will make you feel like you develop / run your apps on a real Android Device.

The skin for the emulator is a folder with graphics. This folder has to be copied in the target platform skins folder of your Android SDK. Due to the fact that some of the skins are high resolution you may need to use the scale option in the Android AVD properties in order to adjust the size of the emulator display.

Downloading new platform skins:

The Google Nexus skins described below can be downloaded from the URL below or scan the barcode on right:

http://heikobehrens.net/2011/03/15/Android-skins/

The package contains four different themes including overlays for a glare effect.

Skins in the package:

- Nexus – ONE
- Nexus – ONE – Black
- Nexus – ONE – Silver
- Nexus – S

Nexus One Screenshots:

HTC-Hero Skin

The HTC Hero skin looks just like the real device. You could download it from:

http://impressive-artworx.de/2011/htc-hero-emulator-skin/

Screenshots:

HTC Touch HD Skin

The HTC Touch HD Smartphone is with a WVGA Display and a resolution of 480x800 pixels

http://www.Android.encke.net/Android-emulator-htc-touchhd-portrait.html

Irdroid AVD skins library

You could download our skin's library collection which contains the skins presented above as well as some other skins of the most popular Android Devices.

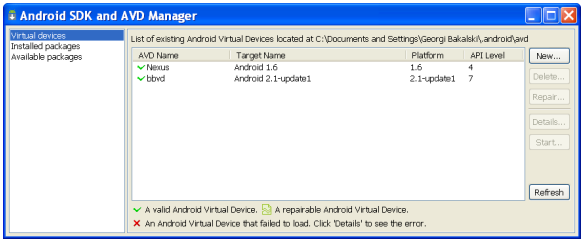
Visit http://www.Irdroid.com/skins/library.zip in order to download our skins library for the Android AVD.

Installing AVD skins.

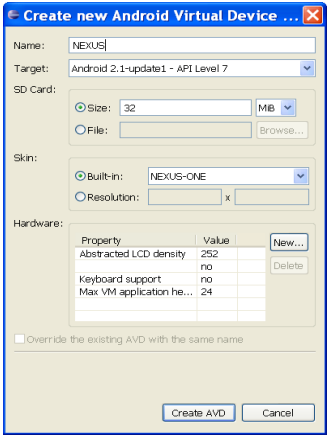
The skins for the Android AVD are folders with Graphics. The installation of the skins is actually quite a simple process. You need to download a skin and decompress the archive. Identify the location of your Android SDK and copy the skin folder to your Android SDK platform folder eg:

```
C:\Android\sdk_r07\Android-sdk-
windows\platforms\Android-7\skins
```

In this case your new skins will be extracted for the Android 7 platform. To setup a new AVD with your newly installed skin you need to start Eclipse, click on “windows” and select “Android SDK and AVD Manager” from the list.



Click “New”



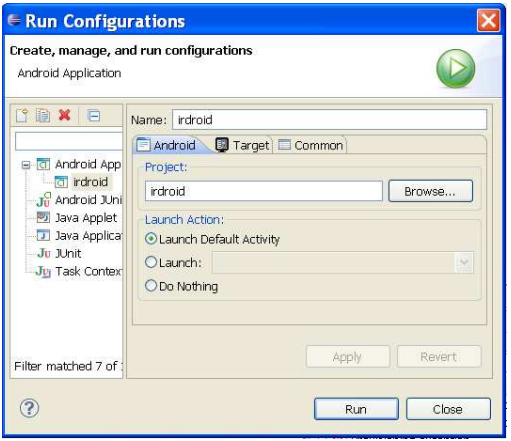
The AVD manager will allow you to select your newly installed skin as well as parameters like name of the new AVD device, SDCARD size in MB, skin for the new AVD.

In the example on the right the NEXUS-ONE skin is selected. You could also change / add additional hardware features for your new AVD. After you are done with your settings, you should click on “Create AVD” in order to save the settings and to create your new AVD device.

Running Irdroid

Starting the Irdroid app for a first time from Eclipse will require defining a run configuration.

Select the Project (Irdroid) and then Click on “Launch” radio button and select the Irdroid Default activity (Irdroid). Afterwards you can click “Run” and start the emulator and the App.



Be patient, the emulator needs some time to start up.

At the end the result should look like this:

…and if you have installed your custom Android skin should look something like the image on the right ->

The App User Interface is very simple. The six buttons are for some basic IR remote functions like Volume, Power, Mute, Channel+ and Channel-. The Remote control device can be selected from the list of available devices from the first

drop-down, the second shows the selected device commands.

Testing Irdroid on the Emulator

It is possible to actually make a physical Irdroid test with the Irdroid app, an Irdroid module connected to your PC. Start your AVD and run the Irdroid app on it. Plug the Irdroid module in your computer phone's 3.5mm audio jack and set the volume level to ½ or ¼. Tap menu->update DB in the AVD in order to update the default list of IR devices. Afterward you can play with the app / your TV set.

Customizing Irdroid source code

As we already mentioned before, the Irdroid application consist of 2 Android activities one java class and the LIRC ported for Android (used as a native library).

The “Lirc.java” class source code

Let’s start with the class called “Lirc.java”

```
--------------------------------------------------------------------------------
----
package com.microcontrollerbg.Irdroid;

import java.io.File;

public class Lirc {

public static String POWER_TOGGLE = "Function18";

static {
        System.loadLibrary ("Irdroid");
       }

native int parse(String filename);

native byte[] getIrBuffer(String irDevice, String irCode);

native String[] getDeviceList();

native String[] getCommandList(String irDevice);

     Lirc (){

               File dir = new File("/mnt/sdcard/log");
               dir.mkdirs();
```

```
    }
}
```

Normally this class loads the LIRC native library called “Irdroid” and you should not change anything here.

The “Iconic” Activity source code

The “Iconic” Activity task is to connect to the Irdroid website, download the default conf file, and load it in the Irdroid application.

```
package com.microcontrollerbg.Irdroid;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;
import org.apache.http.util.ByteArrayBuffer;
import Android.app.*;
import Android.content.Intent;
import Android.os.*;
import Android.widget.Toast;

public class Iconic extends Activity {

    private Handler mHandler;
    Intent intent;
    private final String fileName = "/sdcard/tmp/t.conf";

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
     intent = getIntent();

     mHandler = new Handler();
     checkUpdate.start();
     }

private Thread checkUpdate = new Thread() {
public void run() {
try {
     File file = new File(fileName);
     if (file.exists()) {
     file.delete();
     }
URL updateURL = new
URL("http://www.Irdroid.com/db/t.conf");
URLConnection conn = updateURL.openConnection();
InputStream is = conn.getInputStream();
BufferedInputStream bis = new BufferedInputStream(is);
ByteArrayBuffer baf = new ByteArrayBuffer(50);

     int current = 0;
     while ((current = bis.read()) != -1) {
     baf.append((byte) current);
     }

     File tmpdir = new File("/sdcard/tmp/");

     if (!tmpdir.exists()) {

     tmpdir.mkdirs();
     }

     FileOutputStream fos = new FileOutputStream(file);
     fos.write(baf.toByteArray());

     fos.close();
     setResult(RESULT_OK, intent);
     finish();
     mHandler.post(showUpdate);

     } catch (Exception e) {
}
}
};
```

```
private Runnable showUpdate = new Runnable() {
public void run() {
Toast.makeText(Iconic.this, "Success!",
Toast.LENGTH_SHORT).show();
      }
};
```

The Irdroid Activity source code

The main Irdroid Activity task is to "communicate" with the shared library libIrdroid.so which actually reads the LIRC conf files and provides an interface to the LIRC files.

```
package com.microcontrollerbg.Irdroid;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import Android.app.Activity;
import Android.app.AlertDialog;
import Android.app.Dialog;
import Android.app.AlertDialog.Builder;
import Android.content.Context;
import Android.content.DialogInterface;
import Android.content.Intent;
import Android.content.SharedPreferences;
import Android.media.AudioFormat;
import Android.media.AudioManager;
import Android.media.AudioTrack;
import Android.os.Bundle;
import Android.os.Handler;
import Android.os.SystemClock;
import Android.os.Vibrator;
import Android.text.method.LinkMovementMethod;
import Android.util.Log;
import Android.view.KeyEvent;
import Android.view.Menu;
import Android.view.MenuItem;
import Android.view.MotionEvent;
```

```
import Android.view.View;
import Android.view.View.OnTouchListener;
import Android.widget.AdapterView;
import Android.widget.ArrayAdapter;
import Android.widget.Button;
import Android.widget.EditText;

import Android.widget.Spinner;
import Android.widget.TextView;
import Android.widget.Toast;

/**
 * @author irdroid.com
 *
 */

public class Irdroid extends Activity {
public int number = 0;
byte buffer[];
protected String com;
protected String dev;
SharedPreferences mPrefs;
private AudioManager audio;
public AudioTrack ir;

public int bufSize = AudioTrack.getMinBufferSize(48000,
     AudioFormat.CHANNEL_CONFIGURATION_STEREO,
AudioFormat.ENCODING_PCM_8BIT);

       private final static String LIRCD_CONF_FILE =
       "/sdcard/tmp/t.conf";
private Handler mHandler = new Handler();
public String mycmd;

// global variables
TextView tv;
Lirc lirc;
ArrayAdapter<String> deviceList;
ArrayAdapter<String> commandList;
private Vibrator myVib;
public String gdevice;

private Runnable voldown = new Runnable() {
public void run() {
```

```
ir.release();

String coolcmd = "VOL-";

myVib.vibrate(50);
try {

sendSignal(gdevice, coolcmd);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.postAtTime(this, SystemClock.uptimeMillis() +
250);
}
};
private Runnable volup = new Runnable() {
public void run() {

ir.release();

String coolcmd = "VOL+";

myVib.vibrate(50);

try {

sendSignal(gdevice, coolcmd);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.postAtTime(this, SystemClock.uptimeMillis() +
250);

}
};

private Runnable next = new Runnable() {
public void run() {

ir.release();
```

```
String coolcmd = "P+";

myVib.vibrate(50);

try {

sendSignal(gdevice, coolcmd);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.postAtTime(this, SystemClock.uptimeMillis() +
250);

}
};
private Runnable prev = new Runnable() {
public void run() {

ir.release();

String coolcmd = "P+";

myVib.vibrate(50);

try {

sendSignal(gdevice, coolcmd);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.postAtTime(this, SystemClock.uptimeMillis() +
250);

}
};

public void onCreate(Bundle savedInstanceState) {

audio = (AudioManager)
getSystemService(Context.AUDIO_SERVICE);
```

```
int currentVolume =
audio.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
audio.setStreamVolume(AudioManager.STREAM_MUSIC,
currentVolume / 2, 0);
audio = (AudioManager)
getSystemService(Context.AUDIO_SERVICE);
super.onCreate(savedInstanceState);

firstRunPreferences();
if (getFirstRun()) {
About();
setRunned();

}
setContentView(R.layout.apple);
Button apple_volup = (Button)
findViewById(R.id.apple_volup);
Button apple_voldn = (Button)
findViewById(R.id.apple_voldn);
Button apple_menu = (Button) findViewById(R.id.apple_menu);
Button apple_next = (Button) findViewById(R.id.apple_next);
Button apple_prev = (Button) findViewById(R.id.apple_prev);
Button apple_play = (Button) findViewById(R.id.apple_play);
final Spinner spinDevice = (Spinner)
findViewById(R.id.Spinner01);
final Spinner spinCommand = (Spinner)
findViewById(R.id.Spinner02);
lirc = new Lirc();

myVib = (Vibrator) this.getSystemService(VIBRATOR_SERVICE);

// Initialize adapter for device spinner
deviceList = new ArrayAdapter<String>(this,
Android.R.layout.simple_spinner_item);
deviceList
.setDropDownViewResource(Android.R.layout.simple_spinner_dr
opdown_item);

spinDevice.setPrompt("Select a device");
spinDevice.setAdapter(deviceList);

// Command adapter
commandList = new ArrayAdapter<String>(this,
Android.R.layout.simple_spinner_item);
commandList
```

```
.setDropDownViewResource(Android.R.layout.simple_spinner_dr
opdown_item);

// Parse configuration file and update device adapter
parse(LIRCD_CONF_FILE);

spinDevice
.setOnItemSelectedListener(new
Spinner.OnItemSelectedListener() {
public void onItemSelected(AdapterView<?> parent,
View view, int pos, long id) {
String[] str = lirc.getCommandList(spinDevice
.getSelectedItem().toString());
commandList.clear();
gdevice = spinDevice.getSelectedItem().toString();
for (int i = 0; i < str.length; i++) {
commandList.add(str[i]);
}

}

@Override
public void onNothingSelected(AdapterView<?> arg0) {
}
});

spinCommand.setPrompt("Select a command");
spinCommand.setAdapter(commandList);

spinCommand
.setOnItemSelectedListener(new
Spinner.OnItemSelectedListener() {

@Override
public void onItemSelected(AdapterView<?> parent,
View view, int pos, long id) {

String device = spinDevice.getSelectedItem().toString();
mycmd = spinCommand.getSelectedItem().toString();
if (ir != null) {
try {
sendSignal(device, mycmd);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
```

```
e.printStackTrace();
}
}
}

@Override
public void onNothingSelected(AdapterView<?> arg0) {
// TODO Auto-generated method stub

}

});
apple_voldn.setOnTouchListener(new OnTouchListener() {
public boolean onTouch(View view, MotionEvent motionevent)
{
int action = motionevent.getAction();

if (action == MotionEvent.ACTION_DOWN) {

if (spinDevice.getSelectedItem() == null
|| spinCommand.getSelectedItem() == null) {
Toast.makeText(getApplicationContext(),
"Please select a device and a command",
Toast.LENGTH_SHORT).show();

return true;
}
myVib.vibrate(50);

String mycmd = "VOL-";

try {

sendSignal(gdevice, mycmd);
// Log.i("repeatBtn", "MotionEvent.ACTION_DOWN");
// mHandler.removeCallbacks(mUpdateTask);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

mHandler.postAtTime(voldown,
SystemClock.uptimeMillis() + 250);
}
```

```
else if (action == MotionEvent.ACTION_UP) {
// Log.i("repeatBtn", "MotionEvent.ACTION_UP");\
try {
Thread.sleep(180);
if (ir != null) {
ir.flush();
ir.release();

}
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.removeCallbacks(voldown);

}
return false;
}

});

apple_next.setOnTouchListener(new OnTouchListener() {
public boolean onTouch(View view, MotionEvent motionevent)
{
int action = motionevent.getAction();
if (action == MotionEvent.ACTION_DOWN) {
if (spinDevice.getSelectedItem() == null
|| spinCommand.getSelectedItem() == null) {
Toast.makeText(getApplicationContext(),
"Please select a device and a command",
Toast.LENGTH_SHORT).show();
return true;
}
myVib.vibrate(50);

String gcmd = "P+";

try {
sendSignal(gdevice, gcmd);
Log.i("repeatBtn", "MotionEvent.ACTION_DOWN");
// mHandler.removeCallbacks(mUpdateTask);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
```

```
e.printStackTrace();
}

// mHandler.removeCallbacks(mUpdateTask);

mHandler.postAtTime(next, SystemClock.uptimeMillis() +
250);

} else if (action == MotionEvent.ACTION_UP) {

try {
Thread.sleep(150);
if (ir != null) {
ir.flush();
ir.release();

}
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.removeCallbacks(next);

}
return false;
}
});

apple_prev.setOnTouchListener(new OnTouchListener() {
public boolean onTouch(View view, MotionEvent motionevent)
{
int action = motionevent.getAction();
if (action == MotionEvent.ACTION_DOWN) {
if (spinDevice.getSelectedItem() == null
|| spinCommand.getSelectedItem() == null) {
Toast.makeText(getApplicationContext(),
"Please select a device and a command",
Toast.LENGTH_SHORT).show();
return true;
}
myVib.vibrate(50);

String gcmd = "P-";

try {
```

```
sendSignal(gdevice, gcmd);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

mHandler.postAtTime(prev, SystemClock.uptimeMillis() +
250);

} else if (action == MotionEvent.ACTION_UP) {

try {
Thread.sleep(150);
if (ir != null) {
ir.flush();
ir.release();

}
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.removeCallbacks(prev);

}
return false;
}
});

apple_menu.setOnTouchListener(new OnTouchListener() {
public boolean onTouch(View view, MotionEvent motionevent)
{
int action = motionevent.getAction();
if (action == MotionEvent.ACTION_DOWN) {
if (spinDevice.getSelectedItem() == null
|| spinCommand.getSelectedItem() == null) {
Toast.makeText(getApplicationContext(),
"Please select a device and a command",
Toast.LENGTH_SHORT).show();
return true;
}
myVib.vibrate(50);

String gcmd = "MUTE";
```

```
try {
sendSignal(gdevice, gcmd);

} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

} else if (action == MotionEvent.ACTION_UP) {
try {
Thread.sleep(150);
if (ir != null) {
ir.flush();
ir.release();

}
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

}
return false;
}
});
apple_play.setOnTouchListener(new OnTouchListener() {
public boolean onTouch(View view, MotionEvent motionevent)
{
int action = motionevent.getAction();
if (action == MotionEvent.ACTION_DOWN) {
if (spinDevice.getSelectedItem() == null
|| spinCommand.getSelectedItem() == null) {
Toast.makeText(getApplicationContext(),
"Please select a device and a command",
Toast.LENGTH_SHORT).show();
return true;
}
myVib.vibrate(50);
// String cmd=null;
String gcmd = "POWER";

try {
sendSignal(gdevice, gcmd);
```

```
} catch (IllegalStateException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

Log.i("repeatBtn", "MotionEvent.ACTION_DOWN");

} else if (action == MotionEvent.ACTION_UP) {

try {
Thread.sleep(150);
if (ir != null) {
ir.flush();
ir.release();

}
} catch (InterruptedException e) {

e.printStackTrace();
}

}
return false;

}
});
apple_volup.setOnTouchListener(new OnTouchListener() {

public boolean onTouch(View view, MotionEvent motionevent)
{
int action = motionevent.getAction();

if (action == MotionEvent.ACTION_DOWN) {

if (spinDevice.getSelectedItem() == null
|| spinCommand.getSelectedItem() == null) {
Toast.makeText(getApplicationContext(),
"Please select a device and a command",
Toast.LENGTH_SHORT).show();
return true;
}
myVib.vibrate(50);

String mycmd = "VOL+";
```

```
try {
sendSignal(gdevice, mycmd);

} catch (IllegalStateException e) {

e.printStackTrace();
}

mHandler.postAtTime(volup, SystemClock.uptimeMillis() +
250);

} else if (action == MotionEvent.ACTION_UP) {
// Log.i("repeatBtn", "MotionEvent.ACTION_UP");\
try {
Thread.sleep(150);
if (ir != null) {
ir.flush();
ir.release();

}
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
mHandler.removeCallbacks(volup);

}
return false;

}

});
}

public String selectFile() {

final EditText ed = new EditText(this);

Builder builder = new Builder(this);
builder.setTitle("Select a file to parse");
builder.setView(ed);
builder.setPositiveButton("Save",
new DialogInterface.OnClickListener() {
public void onClick(DialogInterface dialog, int which) {
if (which == Dialog.BUTTON_NEGATIVE) {
```

```
dialog.dismiss();
return;
}
parse(ed.getText().toString());
}
});
builder.setNegativeButton("OK", null);
final AlertDialog asDialog = builder.create();
asDialog.show();
return null;

}

       public String About() {
                AlertDialog.Builder about = new
AlertDialog.Builder(this);
                about.setTitle(R.string.app_name)
                // .setIcon(R.drawable.dialog_icon)
                .setMessage(R.string.info)
                .setCancelable(true)
                .setNegativeButton("Dismiss",
                                new
DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface
dialog, int id) {
                                dialog.dismiss();
                        }
                });

                AlertDialog welcomeAlert = about.create();
                welcomeAlert.show();
                ((TextView)
welcomeAlert.findViewById(Android.R.id.message))

.setMovementMethod(LinkMovementMethod.getInstance());
                return null;

       }

public boolean parse(String config_file) {

java.io.File file = new java.io.File(config_file);

if (!file.exists()) {
if (config_file != LIRCD_CONF_FILE)
```

```
Toast.makeText(getApplicationContext(),
"The Selected file doesn't exist", Toast.LENGTH_SHORT)
.show();
else
Toast.makeText(getApplicationContext(),
"Configuartion file missing, please update the db",
Toast.LENGTH_SHORT).show();
// selectFile();
return false;
}

if (lirc.parse(config_file) == 0) {
Toast.makeText(getApplicationContext(),
"Couldn't parse the selected file", Toast.LENGTH_SHORT)
.show();
// selectFile();
return false;
}

// Save the file since it has been parsed successfully
if (config_file != LIRCD_CONF_FILE) {
try {
FileInputStream in = new FileInputStream(config_file);
FileOutputStream out = new
FileOutputStream(LIRCD_CONF_FILE);
byte[] buf = new byte[1024];
int i = 0;
while ((i = in.read(buf)) != -1) {
out.write(buf, 0, i);
}
in.close();
out.close();
} catch (Exception e) {
tv.append("Probleme saving configuration file: "
+ e.getMessage());
}
}

updateDeviceList();
return true;
}

public void updateDeviceList() {
String[] str = lirc.getDeviceList();
```

```
if (str == null) {
Toast.makeText(getApplicationContext(),
"Invalid, empty or missing config file",
Toast.LENGTH_SHORT)
.show();
// selectFile();
return;
}

deviceList.clear();
for (int i = 0; i < str.length; i++) {
Log.e("ANDRPOLIRC",
String.valueOf(i) + "/" + String.valueOf(str.length) + ": "
+ str[i]);
deviceList.add(str[i]);
}

}

void sendSignal(String device, String cmd) {

buffer = lirc.getIrBuffer(device, cmd);

if (buffer == null) {
Toast.makeText(getApplicationContext(), "Empty Buffer!",
Toast.LENGTH_SHORT).show();
return;
}
ir = new AudioTrack(AudioManager.STREAM_MUSIC, 48000,
AudioFormat.CHANNEL_CONFIGURATION_STEREO,
AudioFormat.ENCODING_PCM_8BIT, bufSize,
AudioTrack.MODE_STATIC);

if (bufSize < buffer.length)
bufSize = buffer.length;
ir.write(buffer, 0, buffer.length);
ir.setStereoVolume(1, 1);

ir.play();
}

void deleteConfigFile() {
java.io.File file = new java.io.File(LIRCD_CONF_FILE);
if (!file.exists())
Toast.makeText(getApplicationContext(),
```

```
"Configuartion file missing\n" + "No file to delete",
Toast.LENGTH_SHORT).show();
else if (file.delete()) {
Toast.makeText(getApplicationContext(),
"File deleted successfully", Toast.LENGTH_SHORT).show();
deviceList.clear();
commandList.clear();

} else
Toast.makeText(getApplicationContext(), "Couldn't delete
the file",
Toast.LENGTH_SHORT).show();
}

public boolean onCreateOptionsMenu(Menu menu) {
menu.add(0, 0, 0, "Parse file").setIcon(
Android.R.drawable.ic_menu_upload);
menu.add(0, 1, 0, "Clear conf").setIcon(
Android.R.drawable.ic_menu_delete);
menu.add(0, 2, 0, "Update
db").setIcon(Android.R.drawable.ic_input_add);
menu.add(0, 3, 0,
"Send").setIcon(Android.R.drawable.arrow_up_float);
menu.add(0, 4, 0,
"About").setIcon(Android.R.drawable.ic_menu_help);
return true;
}

protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
super.onActivityResult(requestCode, resultCode, data);
if (resultCode == RESULT_OK && requestCode == 1) {

parse(LIRCD_CONF_FILE);
updateDeviceList();
}
}

/**
*
* get if this is the first run
*
*
*
* @return returns true, if this is the first run
```

```
*/

public boolean getFirstRun() {

return mPrefs.getBoolean("firstRun", true);

}

/**
*
* store the first run
*/

public void setRunned() {

SharedPreferences.Editor edit = mPrefs.edit();

edit.putBoolean("firstRun", false);

edit.commit();

}

public boolean onKeyDown(int keyCode, KeyEvent event) {
switch (keyCode) {
case KeyEvent.KEYCODE_VOLUME_UP:
audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,
AudioManager.ADJUST_RAISE, AudioManager.FLAG_SHOW_UI);
return true;
case KeyEvent.KEYCODE_VOLUME_DOWN:
audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,
AudioManager.ADJUST_LOWER, AudioManager.FLAG_SHOW_UI);
return true;

default:
super.onKeyDown(keyCode, event);
return false;
}
}

/**
*
* setting up preferences storage
*/
```

```
public void firstRunPreferences() {

Context mContext = this.getApplicationContext();

mPrefs = mContext.getSharedPreferences("myAppPrefs", 0); //
0 = mode
}

public boolean onOptionsItemSelected(MenuItem item) {
switch (item.getItemId()) {
case 0:
selectFile();
break;
case 1:
deleteConfigFile();
break;
case 2:
Intent myIntent = new Intent(Irdroid.this, Iconic.class);
startActivityForResult(myIntent, 1);
break;
case 3:
try {

sendSignal(gdevice, mycmd);

} catch (IllegalStateException e) {

e.printStackTrace();
}
break;
case 4:

About();
break;
}
return false;
}
}
```

Glossary

ADT- Android Development tools

Android Activity – A single focused thing that the user can do.

Android AVD – Android Virtual Device emulator

Android NDK – Google's Android native development kit

Android SDK – Google's Android software development kit

DIY – Do-It-Yourself

Eclipse – IDE (Integrated Development Environment)

GIT – Fast version control system

IDE – Integrate Development Environment

IR – infrared

JAVA – Programming Language

JRE – Java runtime environment

LED – Light emitting diode

LIRC – Linux infrared remote control

PCB – printed circuit board

QR Barcode – Quick response code

SDCARD – Secure Digital Memory card

SMD – Surface mount device / surface mount technology

URI – Uniform resource identifier

XML – Extensible Markup Language

Technical Specifications – Irdroid v.1.0

Irdroid™ v.1.0 Technical Data:

- Operating range > 10 Meters
- IR LED Wavelength – 940 nm
- Hardware Interface Stereo jack 3.5mm
- Amplifier IC – LM386-M1
- Battery type – 4LR44/6V
- Dimensions 43,2x17mm
- Board Thickness – 1mm double layer
- Packing – Carton Box size 60x60x30mm
- Jumper switch for saving power when not in use

Irdroid™ v.1.0 module apps compatibility:

- Irdroid™ – The official Irdroid™ app
- Androlirc – A port of LIRC for Android

- PhotoIrMote

App Specifications:

- Supports Android version 1.6 and above
- Dynamic Layout generation
- Supports LIRC configuration files
- Supports vibration on button press (Heptic feedback)
- A full list of the supported remotes can be found here http://lirc.sourceforge.net/remotes/ .
- Plug and play Design (The user plugs the hardware module and it works)

References:

http://www.irdroid.com – The official Irdroid website
http://www.zokama.com – The Androlirc Application
http://lirc.sf.net – The official LIRC website
http://developer.android.com – Android SDK how-to's
http://www.eclipse.com – The Eclipse IDE website
http://www.github.com/irdroid - Irdroid's GIT repository

About the Author

Irdroid.com - www.irdroid.com


Learn how to:
- Customize the Irdroid source code
- Setup your development computer
- Configure the Android Emulator.
- Make a custom Infrared module.
- Make your own infrared control APP.