**Henning Groß, Beuth Hochschule für Technik Berlin. Studiengang Technische Informatik Bachelor, 749992**

**Anlage zur Bachelorarbeit**

**Programmlistings**

**Henning Groß**

Inhaltsverzeichnis

[1. Programmlistings 2](#_Toc341623963)

[1.1. Messsystem 2](#_Toc341623964)

[1.1.1. VesComIV.c 2](#_Toc341623965)

[1.1.2. EGT.h 4](#_Toc341623966)

[1.1.3. EGT.c 4](#_Toc341623967)

[1.1.4. SerialCommunication.h 6](#_Toc341623968)

[1.1.5. SerialCommunication.c 6](#_Toc341623969)

[1.2. Einheit zur Visualisierung der Messdaten 9](#_Toc341623970)

[1.2.1. layout/vesdroid.xml 9](#_Toc341623971)

[1.2.2. src/VesDroidActivity.java 11](#_Toc341623972)

[1.2.3. src/bluetoothmanagement/BluetoothManagament.java 15](#_Toc341623973)

[1.2.4. src/bluetoothmanagement/MessageHandler.java 19](#_Toc341623974)

[1.2.5. src/packet/VesComPacket.java 22](#_Toc341623975)

[1.2.6. src/packet/VesComPacketHandler.java 24](#_Toc341623976)

[1.2.7. src/packet/LogPacketHandler.java 25](#_Toc341623977)

[1.2.8. src/packet/UpdateUIPacketHandler.java 26](#_Toc341623978)

[1.2.9. src/exceptions/BluetoothNotAvailableException.java 27](#_Toc341623979)

[1.2.10. src/exceptions/BluetoothNotReadyException.java 28](#_Toc341623980)

[1.2.11. src/util/Properties.java 29](#_Toc341623981)

[1.2.12. Android SDK-Klassen 30](#_Toc341623982)

# Programmlistings

## Messsystem

### VesComIV.c

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <avr/eeprom.h>

#include <string.h>

#include <avr/wdt.h>

#include <stdlib.h>

#include "SerialCommunication.h"

#include "EGT.h"

// clock prescaler

#define CLOCK\_PRESCALER 256

// resulting ticks per second of clock

#define TICKS\_PER\_SECOND F\_CPU/CLOCK\_PRESCALER

// resulting ticks per quarter second

#define TICKS\_PER\_QUARTER\_SECOND TICKS\_PER\_SECOND/4

//time of last tacho impulse

volatile uint16\_t v\_old=0;

//v timediff

volatile uint16\_t v\_diff=0;

//the old timestamp of rpm

volatile uint16\_t rpm\_old=0;

//the timediff

volatile uint16\_t rpm\_diff=0;

/\*\*

\* tacho interrupt-routine

\*/

ISR(INT0\_vect){

if(v\_old == 0)

v\_diff = 0;

else

v\_diff = TCNT1 - v\_old;

v\_old = TCNT1;

}

/\*\*

\* rpm interrupt routine

\*/

ISR(INT1\_vect){

if(rpm\_old == 0)

rpm\_diff = 0;

else

rpm\_diff = TCNT1 - rpm\_old;

rpm\_old = TCNT1;

}

/\*\*

\* program-entry with main loop

\*/

int main(void){

// set timer1 (system clock) prescaler to 256

TCCR1B |= (1<<CS12);

// set interrupt-pin interrupts for rpm/v

// v, falling edge

MCUCR |= (1<<ISC01)| (1<<ISC00);

EIMSK |= (1<<INT0);

// rpm, rising edge

MCUCR |= (1<<ISC11) | (1<<ISC10);

EIMSK |= (1<<INT1);

// initialize uart

uart\_init();

// initialize egt

egt\_init();

// enable interrupts

sei();

// last packet send time

int lastPacketSendTime = 0;

// last egt measure time

int lastEGTMeasureTime = 0;

// last egt measurements value

int egtValue = 0;

while(1){

// send a packet every quarter second

if(lastPacketSendTime == 0 || TCNT1 - lastPacketSendTime >= TICKS\_PER\_QUARTER\_SECOND){

sendPacket(rpm\_diff, v\_diff, egtValue);

}

// measure egt every second

if(lastEGTMeasureTime == 0 || TCNT1 - lastEGTMeasureTime >= TICKS\_PER\_SECOND){

egtValue = get\_egt();

}

}

}

### EGT.h

#ifndef EGT\_H\_

#define EGT\_H\_

#define EGT\_PORT PORTC

#define EGT\_PIN PINC

#define EGT\_DATA PINC2

#define EGT\_CS PC1

#define EGT\_CLK PC0

#define EGT\_DELAY 1000

void egt\_init();

uint16\_t get\_egt();

#endif

### EGT.c

#include <avr/io.h>

#include <util/delay.h>

#include "egt.h"

#define EGT\_REGISTER DDRA

/\*\*

\* initializes the ports and pins for egt measurement

\*/

void egt\_init(){

//datenrichtung

DDRC |= (1 << DDC0) | (1 << DDC1);

//cs high for measure

EGT\_PORT |= (1 << EGT\_CS);

//clk low

EGT\_PORT &= ~(1 << EGT\_CLK);

}

/\*\*

\* communciates with max6675 to determine egt

\* all delay-calls are because the max6675 does not support timings as fast as the mega

\*/

uint16\_t get\_egt(){

uint16\_t temp=0;

//cs low for data

EGT\_PORT &= ~(1 << EGT\_CS);

\_delay\_ms(EGT\_DELAY);

//clk high for irrelevant bit

EGT\_PORT |= (1 << EGT\_CLK);

\_delay\_ms(EGT\_DELAY);

//clk low

EGT\_PORT &= ~(1 << EGT\_CLK);

\_delay\_ms(EGT\_DELAY);

//now go through relevant bits and shift them in

for(int i=0; i<10; i++){

//clk high for next bit

EGT\_PORT |= (1 << EGT\_CLK);

\_delay\_ms(EGT\_DELAY);

//shift in bit

temp = temp << (EGT\_PIN & (1 << EGT\_DATA));

//clk low

EGT\_PORT &= ~(1 << EGT\_CLK);

\_delay\_ms(EGT\_DELAY);

}

return temp;

}

### SerialCommunication.h

void uart\_init();

void uart\_putc(unsigned char c);

void uart\_puts (char \*s);

void sendPacket(int rpm\_diff, int v\_diff, int egt);

int addIntToString(char\* existingString, int offset, int addMe);

### SerialCommunication.c

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <string.h>

#include <stdlib.h>

#include "SerialCommunication.h"

#define FOSC 8000000// Clock Speed

#define BAUD 19200 //gps

#define MYUBRR (FOSC/16/BAUD-1)

int packetNumber = 0;

/\*\*

\* increases packet number and resets it when it would soon overflow otherwise

\*/

int getPacketNumber(){

if(packetNumber>60000){

packetNumber = 0;

}

packetNumber++;

return packetNumber;

}

/\*\*

\* initializes the uart

\*/

void uart\_init(){

UCSR0B = (1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0);

UCSR0C |= (3<<UCSZ00); // asynchron 8N1

UBRR0H = (unsigned char)(MYUBRR>>8);

UBRR0L = (unsigned char)MYUBRR;

}

/\*\*

\* sends a single character via uart

\*/

void uart\_putc(unsigned char c)

{

// wait for uart to be ready

while (!(UCSR0A & (1<<UDRE0))){

}

// put character in output register

UDR0 = c;

// wait for uart to be ready again (means character was sent)

while (!(UCSR0A & (1<<UDRE0))){

}

}

/\*\*

\* sends a character array via uart

\*/

void uart\_puts (char \*s)

{

while (\*s)

{

// simply putc for each character in the array/string

uart\_putc(\*s);

s++;

}

}

/\*\*

\* adds a string representation of a given integer to a string at the given offset

\* make sure that the given integer is not longer than 10 digits at a base of 10

\* and that the string that we append to has enough reserved memory

\* returns the number of digits added

\*/

int addIntToString(char\* existingString, int offset, int addMe){

// stringbuffer

char stringbuffer[10];

// convert to string

itoa(addMe, stringbuffer, 10);

// iterator as pointer to the currently handled digit

int iterator = 0;

while(stringbuffer[iterator] != '\0'){

// set digit to existing string

existingString[iterator + offset] = stringbuffer[iterator];

// increase iterator

iterator++;

}

// as the iterator get increased after each successful digit and also before end of string

// it already contains the exact number of digits added

return iterator;

}

/\*\*

\* sends a vescompacket via uart from given values

\*/

void sendPacket(int rpm\_diff, int v\_diff, int egt){

char message[29];

int pointer = 0;

message[pointer] = '\*';

pointer++;

pointer += addIntToString(message, pointer, getPacketNumber());

message[pointer] = ',';

pointer++;

pointer += addIntToString(message, pointer, rpm\_diff);

message[pointer] = ',';

pointer++;

pointer += addIntToString(message, pointer, v\_diff);

message[pointer] = ',';

pointer++;

pointer += addIntToString(message, pointer, egt);

message[pointer] = '#';

message[pointer+1] = '\0';

uart\_puts(message);

}

## Einheit zur Visualisierung der Messdaten

### layout/vesdroid.xml

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="horizontal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="rpm: "

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

<TextView android:id="@+id/rpm"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="n/a"

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

</LinearLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="horizontal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="egt: "

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

<TextView android:id="@+id/egt"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="n/a"

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

</LinearLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="horizontal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="v: "

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

<TextView android:id="@+id/v"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="n/a"

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

</LinearLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="horizontal"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="p#: "

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

<TextView android:id="@+id/packetnumber"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="n/a"

android:background="#666"

android:textColor="#fff"

android:textSize="18pt"

/>

</LinearLayout>

</LinearLayout>

### src/VesDroidActivity.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.widget.TextView;

import android.widget.Toast;

import com.example.android.BluetoothChat.DeviceListActivity;

import com.example.android.BluetoothChat.R;

import de.gaffa.vescom.vesdroid.bluetoothmanagement.BluetoothManagement;

import de.gaffa.vescom.vesdroid.bluetoothmanagement.MessageHandler;

import de.gaffa.vescom.vesdroid.exceptions.BluetoothNotAvailableException;

import de.gaffa.vescom.vesdroid.packet.LogPacketHandler;

import de.gaffa.vescom.vesdroid.packet.UpdateUIPacketHandler;

import de.gaffa.vescom.vesdroid.util.Properties;

/\*\*

\* the entry class representing the main android activity

\*/

public class VesDroidActivity extends Activity {

// management instance for bluetooth-operations

private BluetoothManagement bluetoothManagement;

/\*\*

\* {@inheritDoc}

\*/

@Override

public void onCreate(Bundle savedInstanceState) {

/\*\*

\* "standard" android activity-initialization

\*/

super.onCreate(savedInstanceState);

setContentView(R.layout.vesdroid);

/\*\*

\* find activities textviews

\*/

TextView v = (TextView) findViewById(R.id.v);

TextView rpm = (TextView) findViewById(R.id.rpm);

TextView egt = (TextView) findViewById(R.id.egt);

TextView packetNumber = (TextView) findViewById(R.id.packetnumber);

// initialize Messagehandler-instance

MessageHandler messageHandler = new MessageHandler(15625, 2, 133);

// register handlers for vescom-packets that are received

messageHandler.withVesComPacketHandler(new LogPacketHandler());

messageHandler.withVesComPacketHandler(new UpdateUIPacketHandler(rpm, v, egt, packetNumber));

try {

// initialize/instantiate bluetooth-management

bluetoothManagement = new BluetoothManagement(messageHandler);

} catch (BluetoothNotAvailableException b) {

/\*\*

\* handle devices on which bluetooth is not available

\*/

Toast.makeText(this, "Bluetooth is not available on this device", Toast.LENGTH\_LONG).show();

finish();

return;

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void onStart() {

super.onStart();

bluetoothManagement.onStart(this);

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public synchronized void onResume() {

super.onResume();

bluetoothManagement.onResume();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public synchronized void onPause() {

super.onPause();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void onStop() {

super.onStop();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void onDestroy() {

super.onDestroy();

bluetoothManagement.onDestroy();

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void onActivityResult(int requestCode, int resultCode, Intent data) {

/\*\*

\* handles messages from other activities

\*/

switch (requestCode) {

case Properties.REQUEST\_CONNECT\_DEVICE:

// deviceListActivity from com.example.android.BluetoothChat returns with a device to connect

if (resultCode == Activity.RESULT\_OK) {

// get address

String address = data.getExtras().getString(DeviceListActivity.EXTRA\_DEVICE\_ADDRESS);

// connect

bluetoothManagement.connectDevice(address);

}

break;

case Properties.REQUEST\_ENABLE\_BT:

// the request to enable bluetooth returns

if (resultCode == Activity.RESULT\_OK) {

// bluetooth enabled, setup connection

bluetoothManagement.setupConnection();

} else {

// bluetooth still off, exit

Toast.makeText(this, R.string.bt\_not\_enabled\_leaving, Toast.LENGTH\_SHORT).show();

finish();

}

}

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public boolean onCreateOptionsMenu(Menu menu) {

/\*\*

\* creates options menu

\*/

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.option\_menu, menu);

return true;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public boolean onOptionsItemSelected(MenuItem item) {

/\*\*

\* handles device selection on options menu

\*/

Intent serverIntent = null;

switch (item.getItemId()) {

case R.id.connect\_scan:

// Launch the DeviceListActivity to see devices and do scan

serverIntent = new Intent(this, DeviceListActivity.class);

startActivityForResult(serverIntent, Properties.REQUEST\_CONNECT\_DEVICE);

return true;

}

return false;

}

}

### src/bluetoothmanagement/BluetoothManagament.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.bluetoothmanagement;

import android.app.Activity;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.content.Intent;

import com.example.android.BluetoothChat.BluetoothService;

import de.gaffa.vescom.vesdroid.exceptions.BluetoothNotAvailableException;

import de.gaffa.vescom.vesdroid.exceptions.BluetoothNotReadyException;

import de.gaffa.vescom.vesdroid.util.Properties;

/\*\*

\* this class is loosely based on the BluetoothChat-class that can be found in the sdk-examples

\* and serves as a facade for the vesdroid-app to the (modified) bluetoothchat-BluetoothService

\*/

public class BluetoothManagement {

// the devices bluetooth adapter

private BluetoothAdapter bluetoothAdapter;

// instance of sdk examples bluetoothservice for basic bluetooth-interaction

private BluetoothService bluetoothService;

// bluetooth-messagehandler

private MessageHandler messageHandler;

/\*\*

\* initializes everything nessecary to set up a bluetooth connection

\*

\* @param messageHandler handler-instance that takes care of messages received from bluetooth

\* @throws BluetoothNotAvailableException if the device does not support bluetooth

\*/

public BluetoothManagement(MessageHandler messageHandler) throws BluetoothNotAvailableException {

// access bluetooth adapter

bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

this.messageHandler = messageHandler;

if (bluetoothAdapter == null) {

// no bluetooth adapter found

throw new BluetoothNotAvailableException("This device does not support bluetooth.");

}

}

/\*\*

\* call this on your activities onStart-event and pass the activity

\* this will enable bluetooth if it isnt yet.

\* if you dont do this, exceptions will occur later

\*

\* @param activity the activity that shall receive the notification after the bluetooth-enable-request

\*/

public void onStart(Activity activity) {

if (!bluetoothAdapter.isEnabled()) {

// bluetooth off, try and enable

Intent enableIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

activity.startActivityForResult(enableIntent, Properties.REQUEST\_ENABLE\_BT);

} else {

if (bluetoothService == null) {

// set up bluetooth

setupConnection();

}

}

}

/\*\*

\* initializes bluetooth-service

\*/

public void setupConnection() {

bluetoothService = new BluetoothService(messageHandler);

}

/\*\*

\* check if service is ready to send a message

\*

\* @return true if ready

\*/

public Boolean isReadyToSend() {

return bluetoothService != null && BluetoothService.STATE\_CONNECTED == bluetoothService.getState();

}

/\*\*

\* sends a given message to the coupled bluetooth-device (if any)

\*

\* @param message the message to send

\* @throws BluetoothNotReadyException if bluetooth is not ready to send (check using isReadyToSend() first)

\*/

public void sendMessage(String message) throws BluetoothNotReadyException {

if (!isReadyToSend()) {

// not ready

throw new BluetoothNotReadyException("Not ready to send");

}

// send message

bluetoothService.write(message.getBytes());

}

/\*\*

\* call this method on your activies onResume-event

\* it will start the service if it was dropped meanwhile

\*/

public void onResume() {

if (bluetoothService != null) {

if (bluetoothService.getState() == BluetoothService.STATE\_NONE) {

// (re-)start service

bluetoothService.start();

}

}

}

/\*\*

\* call this method on your activities onDestroy-event

\* it will stop the bluetooth-service

\*/

public void onDestroy() {

if (bluetoothService != null) {

// stop bluetooth-service

bluetoothService.stop();

}

}

/\*\*

\* connects a device

\*

\* @param address device address

\*/

public void connectDevice(String address) {

// get device

BluetoothDevice device = bluetoothAdapter.getRemoteDevice(address);

// connect

bluetoothService.connect(device);

}

}

### src/bluetoothmanagement/MessageHandler.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.bluetoothmanagement;

import android.os.Handler;

import android.os.Message;

import android.util.Log;

import de.gaffa.vescom.vesdroid.packet.VesComPacket;

import de.gaffa.vescom.vesdroid.packet.VesComPacketHandler;

import de.gaffa.vescom.vesdroid.util.Properties;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* handles bluetooth messages and transforms them into vesdroid-packets

\* also calls their handlers

\*/

public class MessageHandler extends Handler {

private final static char START = '\*';

private final static char END = '#';

private final static String SEPARATOR = ",";

// clock ticks per second of the controller

private Integer controllerTicksPerSecond;

// ignition signals to controller per round

private Integer ignitionSignalsPerRound;

// the wheels circumference

private Integer wheelCircumference;

// packet handlers to notify

private List<VesComPacketHandler> vesComPacketHandlers = new ArrayList<VesComPacketHandler>();

// packets are build here until they are fully received

private String messageBuffer;

/\*\*

\* initializes the handler

\*

\* @param controllerTicksPerSecond clock ticks per second of the controller

\* @param ignitionSignalsPerRound ignition signals to controller per round

\* @param wheelCircumference the wheels circumference

\*/

public MessageHandler(Integer controllerTicksPerSecond, Integer ignitionSignalsPerRound, Integer wheelCircumference) {

this.controllerTicksPerSecond = controllerTicksPerSecond;

this.ignitionSignalsPerRound = ignitionSignalsPerRound;

this.wheelCircumference = wheelCircumference;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public void handleMessage(Message msg) {

// handle message received event

if (msg.what == Properties.MESSAGE\_READ) {

byte[] readBuf = (byte[]) msg.obj;

// construct a string from the valid bytes in the buffer

String readMessage = new String(readBuf, 0, msg.arg1);

//mConversationArrayAdapter.add(mConnectedDeviceName + ": " + readMessage);

for (char c : readMessage.toCharArray()) {

if (c == START) {

messageBuffer = new String();

} else if (c == END) {

handlePacket();

} else if (messageBuffer != null) {

messageBuffer += c;

}

}

}

}

/\*\*

\* calculates rpm from diff value

\*

\* @param diff how many ticks since last measure

\* @return

\*/

private Integer convertDiffToRpm(Integer diff) {

return controllerTicksPerSecond / diff / ignitionSignalsPerRound \* 60;

}

/\*\*

\* calculates velocity from diff value

\*

\* @param diff how many ticks since last measure

\* @return

\*/

private Integer convertDiffToV(Integer diff) {

return controllerTicksPerSecond / diff / 3600 \* wheelCircumference / 10 ^ 5;

}

/\*\*

\* converts a bluetooth-message to a vesdroid-packet and notifies all handlers

\*/

private void handlePacket() {

// create value-array from string by splitting at the separator-character

String[] fields = messageBuffer.split(SEPARATOR);

try {

if (fields.length != 4) {

// message seems to be invalid as 4 values are expected

throw new Exception("Message contained more or less than 4 fields");

}

/\*\*

\* assign values

\*/

Integer packetNumber = Integer.parseInt(fields[0]);

Integer rpm = convertDiffToRpm(Integer.parseInt(fields[1]));

Integer v = convertDiffToV(Integer.parseInt(fields[2]));

Integer egt = Integer.parseInt(fields[3]);

// create packet with values

VesComPacket packet = new VesComPacket(packetNumber, rpm, v, egt);

/\*\*

\* notify all registered handlers

\*/

for (VesComPacketHandler vesComPacketHandler : vesComPacketHandlers) {

/\*\*

\* just put entries in the log if anything goes wrong with a single handler. we still want the other

\* handlers to get notified

\*/

try{

vesComPacketHandler.handlePacket(packet);

} catch (Throwable th){

Log.e("VesDroid", "The handler: " + vesComPacketHandler + " caused an Exception: ", th);

}

}

} catch (Throwable tr) {

// log all exceptions that occur (even

Log.e("VesDroid", "Fatal. A Message could not be converted to a VesDroid-Packet.", tr);

}

}

/\*\*

\* builder-style add-accessor for a packet handler

\*

\* @param vesComPacketHandler the handler to register

\* @return the instance itself (builder-style)

\*/

public MessageHandler withVesComPacketHandler(VesComPacketHandler vesComPacketHandler) {

vesComPacketHandlers.add(vesComPacketHandler);

return this;

}

}

### src/packet/VesComPacket.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.packet;

/\*\*

\* representation of a packet received from measure-unit

\*/

public class VesComPacket {

// unique packet identifier

private Integer packetNumber;

// rounds per minute

private Integer rpm;

// velocity

private Integer v;

// exhaust gas temperature

private Integer egt;

/\*\*

\* representation of a packet received from measure-unit

\*

\* @param packetNumber unique packet identifier

\* @param rpm rounds per minute

\* @param v velocity

\* @param egt exhaust gas temperature

\*/

public VesComPacket(Integer packetNumber, Integer rpm, Integer v, Integer egt) {

this.packetNumber = packetNumber;

this.rpm = rpm;

this.v = v;

this.egt = egt;

}

/\*\*

\* getter for packet number

\*

\* @return unique packet identifier

\*/

public Integer getPacketNumber() {

return packetNumber;

}

/\*\*

\* getter for rpm

\*

\* @return rounds per minute

\*/

public Integer getRpm() {

return rpm;

}

/\*\*

\* getter for v

\*

\* @return velocity

\*/

public Integer getV() {

return v;

}

/\*\*

\* getter for egt

\*

\* @return exhaust gas temperature

\*/

public Integer getEgt() {

return egt;

}

/\*\*

\* {@inheritDoc}

\*/

@Override

public String toString() {

return "VesCom Packet with number: " + packetNumber + ". rpm: " + rpm + ". v: " + v + ". egt: " + egt + ".";

}

}

### src/packet/VesComPacketHandler.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.packet;

/\*\*

\* a handler for received VesComPackets

\*/

public interface VesComPacketHandler {

/\*\*

\* handles a VesComPacket

\*

\* @param vesComPacket the packet to handle

\*/

public void handlePacket(VesComPacket vesComPacket);

}

### src/packet/LogPacketHandler.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.packet;

import android.util.Log;

/\*\*

\* {@inheritDoc}

\* logs at debug-level with tag "VesDroid"

\*/

public class LogPacketHandler implements VesComPacketHandler {

/\*\*

\* {@inheritDoc}

\* logs at debug-level with tag "VesDroid"

\*/

@Override

public void handlePacket(VesComPacket vesComPacket) {

Log.d("VesDroid", "Packet received: " + vesComPacket);

}

}

### src/packet/UpdateUIPacketHandler.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.packet;

import android.widget.TextView;

/\*\*

\* {@inheritDoc}

\* updates the ui´s textviews

\*/

public class UpdateUIPacketHandler implements VesComPacketHandler {

private TextView rpmView;

private TextView vView;

private TextView egtView;

private TextView packetNumberView;

/\*\*

\* handler that updates the ui´s textviews

\*

\* @param rpmView the view representing rounds per minute

\* @param vView the view representing velocity

\* @param egtView the view representing exhaust gas temperature

\* @param packetNumberView the view representing the packet number

\*/

public UpdateUIPacketHandler(TextView rpmView, TextView vView, TextView egtView, TextView packetNumberView) {

this.rpmView = rpmView;

this.egtView = egtView;

this.vView = vView;

this.packetNumberView = packetNumberView;

}

/\*\*

\* {@inheritDoc}

\* updates the ui´s textviews

\*/

@Override

public void handlePacket(VesComPacket vesComPacket) {

/\*\*

\* set views values

\*/

rpmView.setText(vesComPacket.getRpm().toString());

egtView.setText(vesComPacket.getEgt().toString());

vView.setText(vesComPacket.getV().toString());

packetNumberView.setText(vesComPacket.getPacketNumber().toString());

}

}

### src/exceptions/BluetoothNotAvailableException.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.exceptions;

/\*\*

\* thrown when the device does not support bluetooth

\*/

public class BluetoothNotAvailableException extends Exception {

/\*\*

\* {@inheritDoc}

\*/

public BluetoothNotAvailableException(String detailMessage) {

super(detailMessage);

}

}

### src/exceptions/BluetoothNotReadyException.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.exceptions;

/\*\*

\* thrown if an action is requested but bluetooth is not ready to fulfill it

\*/

public class BluetoothNotReadyException extends Exception {

/\*\*

\* {@inheritDoc}

\*/

public BluetoothNotReadyException(String detailMessage) {

super(detailMessage);

}

}

### src/util/Properties.java

/\*

\* Copyright (C) 2012 Henning Groß

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package de.gaffa.vescom.vesdroid.util;

/\*\*

\* holds properties for application

\*/

public class Properties {

// application id

public final static String ApplicationID = "VesDroid";

/\*\*

\* message types for/from sdks bluetooth-service

\*/

public static final int MESSAGE\_STATE\_CHANGE = 1;

public static final int MESSAGE\_READ = 2;

public static final int MESSAGE\_WRITE = 3;

public static final int MESSAGE\_DEVICE\_NAME = 4;

public static final int MESSAGE\_TOAST = 5;

public static final String DEVICE\_NAME = "device\_name";

public static final String TOAST = "toast";

/\*\*

\* intent request codes

\*/

public static final int REQUEST\_CONNECT\_DEVICE = 1;

public static final int REQUEST\_ENABLE\_BT = 2;

}

### Android SDK-Klassen

Die beiden hier aufgeführten Klassen wurden im Rahmen dieser Arbeit lediglich modifiziert. Sie stammen aus dem Bluetooth-Chat Beispiel des Android SDK.

#### BluetoothService.java

/\*

\* Copyright (C) 2009 The Android Open Source Project

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package com.example.android.BluetoothChat;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.bluetooth.BluetoothSocket;

import android.os.Bundle;

import android.os.Handler;

import android.os.Message;

import android.util.Log;

import de.gaffa.vescom.vesdroid.util.Properties;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.UUID;

/\*\*

\* This class does all the work for setting up and managing Bluetooth

\* connections with other devices. It has a thread that listens for

\* incoming connections, a thread for connecting with a device, and a

\* thread for performing data transmissions when connected.

\* <p/>

\* This class was heavily modified for the VesDroid-Project

\* If you want to use it for another project please consider using the original AOSP-class instead

\*/

public class BluetoothService {

private static final UUID SERIAL\_UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

private final BluetoothAdapter mAdapter;

private final Handler mHandler;

private ConnectThread mConnectThread;

private ConnectedThread mConnectedThread;

private int mState;

// Constants that indicate the current connection state

public static final int STATE\_NONE = 0; // we're doing nothing

public static final int STATE\_LISTEN = 1; // now listening for incoming connections

public static final int STATE\_CONNECTING = 2; // now initiating an outgoing connection

public static final int STATE\_CONNECTED = 3; // now connected to a remote device

/\*\*

\* Constructor. Prepares a new BluetoothChat session.

\*

\* @param handler A Handler to send messages back to the UI Activity

\*/

public BluetoothService(Handler handler) {

mAdapter = BluetoothAdapter.getDefaultAdapter();

mState = STATE\_NONE;

mHandler = handler;

}

/\*\*

\* Set the current state of the chat connection

\*

\* @param state An integer defining the current connection state

\*/

private synchronized void setState(int state) {

Log.d(Properties.ApplicationID, "setState() " + mState + " -> " + state);

mState = state;

// Give the new state to the Handler so the UI Activity can update

mHandler.obtainMessage(Properties.MESSAGE\_STATE\_CHANGE, state, -1).sendToTarget();

}

/\*\*

\* Return the current connection state.

\*/

public synchronized int getState() {

return mState;

}

/\*\*

\* Start the chat service. Specifically start AcceptThread to begin a

\* session in listening (server) mode. Called by the Activity onResume()

\*/

public synchronized void start() {

Log.d(Properties.ApplicationID, "start");

// Cancel any thread attempting to make a connection

if (mConnectThread != null) {

mConnectThread.cancel();

mConnectThread = null;

}

// Cancel any thread currently running a connection

if (mConnectedThread != null) {

mConnectedThread.cancel();

mConnectedThread = null;

}

setState(STATE\_LISTEN);

}

/\*\*

\* Start the ConnectThread to initiate a connection to a remote device.

\*

\* @param device The BluetoothDevice to connect

\*/

public synchronized void connect(BluetoothDevice device) {

Log.d(Properties.ApplicationID, "connect to: " + device);

// Cancel any thread attempting to make a connection

if (mState == STATE\_CONNECTING) {

if (mConnectThread != null) {

mConnectThread.cancel();

mConnectThread = null;

}

}

// Cancel any thread currently running a connection

if (mConnectedThread != null) {

mConnectedThread.cancel();

mConnectedThread = null;

}

// Start the thread to connect with the given device

mConnectThread = new ConnectThread(device);

mConnectThread.start();

setState(STATE\_CONNECTING);

}

/\*\*

\* Start the ConnectedThread to begin managing a Bluetooth connection

\*

\* @param socket The BluetoothSocket on which the connection was made

\* @param device The BluetoothDevice that has been connected

\*/

public synchronized void connected(BluetoothSocket socket, BluetoothDevice

device, final String socketType) {

Log.d(Properties.ApplicationID, "connected, Socket Type:" + socketType);

// Cancel the thread that completed the connection

if (mConnectThread != null) {

mConnectThread.cancel();

mConnectThread = null;

}

// Cancel any thread currently running a connection

if (mConnectedThread != null) {

mConnectedThread.cancel();

mConnectedThread = null;

}

// Start the thread to manage the connection and perform transmissions

mConnectedThread = new ConnectedThread(socket, socketType);

mConnectedThread.start();

// Send the name of the connected device back to the UI Activity

Message msg = mHandler.obtainMessage(Properties.MESSAGE\_DEVICE\_NAME);

Bundle bundle = new Bundle();

bundle.putString(Properties.DEVICE\_NAME, device.getName());

msg.setData(bundle);

mHandler.sendMessage(msg);

setState(STATE\_CONNECTED);

}

/\*\*

\* Stop all threads

\*/

public synchronized void stop() {

Log.d(Properties.ApplicationID, "stop");

if (mConnectThread != null) {

mConnectThread.cancel();

mConnectThread = null;

}

if (mConnectedThread != null) {

mConnectedThread.cancel();

mConnectedThread = null;

}

setState(STATE\_NONE);

}

/\*\*

\* Write to the ConnectedThread in an unsynchronized manner

\*

\* @param out The bytes to write

\* @see ConnectedThread#write(byte[])

\*/

public void write(byte[] out) {

// Create temporary object

ConnectedThread r;

// Synchronize a copy of the ConnectedThread

synchronized (this) {

if (mState != STATE\_CONNECTED) return;

r = mConnectedThread;

}

// Perform the write unsynchronized

r.write(out);

}

/\*\*

\* Indicate that the connection attempt failed and notify the UI Activity.

\*/

private void connectionFailed() {

// Send a failure message back to the Activity

Message msg = mHandler.obtainMessage(Properties.MESSAGE\_TOAST);

Bundle bundle = new Bundle();

bundle.putString(Properties.TOAST, "Unable to connect device");

msg.setData(bundle);

mHandler.sendMessage(msg);

// Start the service over to restart listening mode

BluetoothService.this.start();

}

/\*\*

\* Indicate that the connection was lost and notify the UI Activity.

\*/

private void connectionLost() {

// Send a failure message back to the Activity

Message msg = mHandler.obtainMessage(Properties.MESSAGE\_TOAST);

Bundle bundle = new Bundle();

bundle.putString(Properties.TOAST, "Device connection was lost");

msg.setData(bundle);

mHandler.sendMessage(msg);

// Start the service over to restart listening mode

BluetoothService.this.start();

}

/\*\*

\* This thread runs while attempting to make an outgoing connection

\* with a device. It runs straight through; the connection either

\* succeeds or fails.

\*/

private class ConnectThread extends Thread {

private final BluetoothSocket mmSocket;

private final BluetoothDevice mmDevice;

private String mSocketType;

public ConnectThread(BluetoothDevice device) {

mmDevice = device;

BluetoothSocket tmp = null;

mSocketType = "Insecure";

// Get a BluetoothSocket for a connection with the

// given BluetoothDevice

try {

tmp = device.createInsecureRfcommSocketToServiceRecord(

SERIAL\_UUID);

} catch (IOException e) {

Log.e(Properties.ApplicationID, "Socket Type: " + mSocketType + "create() failed", e);

}

mmSocket = tmp;

}

public void run() {

Log.i(Properties.ApplicationID, "BEGIN mConnectThread SocketType:" + mSocketType);

setName("ConnectThread" + mSocketType);

// Always cancel discovery because it will slow down a connection

mAdapter.cancelDiscovery();

// Make a connection to the BluetoothSocket

try {

// This is a blocking call and will only return on a

// successful connection or an exception

mmSocket.connect();

} catch (IOException e) {

// Close the socket

try {

mmSocket.close();

} catch (IOException e2) {

Log.e(Properties.ApplicationID, "unable to close() " + mSocketType +

" socket during connection failure", e2);

}

connectionFailed();

return;

}

// Reset the ConnectThread because we're done

synchronized (BluetoothService.this) {

mConnectThread = null;

}

// Start the connected thread

connected(mmSocket, mmDevice, mSocketType);

}

public void cancel() {

try {

mmSocket.close();

} catch (IOException e) {

Log.e(Properties.ApplicationID, "close() of connect " + mSocketType + " socket failed", e);

}

}

}

/\*\*

\* This thread runs during a connection with a remote device.

\* It handles all incoming and outgoing transmissions.

\*/

private class ConnectedThread extends Thread {

private final BluetoothSocket mmSocket;

private final InputStream mmInStream;

private final OutputStream mmOutStream;

public ConnectedThread(BluetoothSocket socket, String socketType) {

Log.d(Properties.ApplicationID, "create ConnectedThread: " + socketType);

mmSocket = socket;

InputStream tmpIn = null;

OutputStream tmpOut = null;

// Get the BluetoothSocket input and output streams

try {

tmpIn = socket.getInputStream();

tmpOut = socket.getOutputStream();

} catch (IOException e) {

Log.e(Properties.ApplicationID, "temp sockets not created", e);

}

mmInStream = tmpIn;

mmOutStream = tmpOut;

}

public void run() {

Log.i(Properties.ApplicationID, "BEGIN mConnectedThread");

byte[] buffer = new byte[1024];

int bytes;

// Keep listening to the InputStream while connected

while (true) {

try {

// Read from the InputStream

bytes = mmInStream.read(buffer);

// Send the obtained bytes to the UI Activity

mHandler.obtainMessage(Properties.MESSAGE\_READ, bytes, -1, buffer)

.sendToTarget();

} catch (IOException e) {

Log.e(Properties.ApplicationID, "disconnected", e);

connectionLost();

break;

}

}

}

/\*\*

\* Write to the connected OutStream.

\*

\* @param buffer The bytes to write

\*/

public void write(byte[] buffer) {

try {

mmOutStream.write(buffer);

// Share the sent message back to the UI Activity

mHandler.obtainMessage(Properties.MESSAGE\_WRITE, -1, -1, buffer)

.sendToTarget();

} catch (IOException e) {

Log.e(Properties.ApplicationID, "Exception during write", e);

}

}

public void cancel() {

try {

mmSocket.close();

} catch (IOException e) {

Log.e(Properties.ApplicationID, "close() of connect socket failed", e);

}

}

}

}

#### DeviceListActivity.java

/\*

\* Copyright (C) 2009 The Android Open Source Project

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package com.example.android.BluetoothChat;

import android.app.Activity;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.view.Window;

import android.widget.AdapterView;

import android.widget.AdapterView.OnItemClickListener;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.TextView;

import java.util.Set;

/\*\*

\* This Activity appears as a dialog. It lists any paired devices and

\* devices detected in the area after discovery. When a device is chosen

\* by the user, the MAC address of the device is sent back to the parent

\* Activity in the result Intent.

\* <p/>

\* This class was heavily modified for the VesDroid-Project

\* If you want to use it for another project please consider using the original AOSP-class instead

\*/

public class DeviceListActivity extends Activity {

// Return Intent extra

public static String EXTRA\_DEVICE\_ADDRESS = "device\_address";

// Member fields

private BluetoothAdapter mBtAdapter;

private ArrayAdapter<String> mPairedDevicesArrayAdapter;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

// Setup the window

requestWindowFeature(Window.FEATURE\_INDETERMINATE\_PROGRESS);

setContentView(R.layout.device\_list);

// Set result CANCELED incase the user backs out

setResult(Activity.RESULT\_CANCELED);

// Initialize array adapters. One for already paired devices and

// one for newly discovered devices

mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this, R.layout.device\_name);

// Find and set up the ListView for paired devices

ListView pairedListView = (ListView) findViewById(R.id.paired\_devices);

pairedListView.setAdapter(mPairedDevicesArrayAdapter);

pairedListView.setOnItemClickListener(mDeviceClickListener);

// Get the local Bluetooth adapter

mBtAdapter = BluetoothAdapter.getDefaultAdapter();

// Get a set of currently paired devices

Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();

// If there are paired devices, add each one to the ArrayAdapter

if (pairedDevices.size() > 0) {

findViewById(R.id.title\_paired\_devices).setVisibility(View.VISIBLE);

for (BluetoothDevice device : pairedDevices) {

mPairedDevicesArrayAdapter.add(device.getName() + "\n" + device.getAddress());

}

} else {

String noDevices = getResources().getText(R.string.none\_paired).toString();

mPairedDevicesArrayAdapter.add(noDevices);

}

}

@Override

protected void onDestroy() {

super.onDestroy();

}

// The on-click listener for all devices in the ListViews

private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {

public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {

// Cancel discovery because it's costly and we're about to connect

mBtAdapter.cancelDiscovery();

// Get the device MAC address, which is the last 17 chars in the View

String info = ((TextView) v).getText().toString();

String address = info.substring(info.length() - 17);

// Create the result Intent and include the MAC address

Intent intent = new Intent();

intent.putExtra(EXTRA\_DEVICE\_ADDRESS, address);

// Set result and finish this Activity

setResult(Activity.RESULT\_OK, intent);

finish();

}

};

}