```
 1  package course.examples.intelligentrobot.activities;
 2
 3  import android.content.Context;
 4  import android.content.Intent;
 5  import android.hardware.usb.UsbDevice;
 6  import android.hardware.usb.UsbDeviceConnection;
 7  import android.hardware.usb.UsbManager;
 8  import android.media.MediaPlayer;
 9  import android.net.nsd.NsdManager;
10  import android.net.nsd.NsdServiceInfo;
11  import android.os.Bundle;
12  import android.os.Handler;
13  import android.speech.tts.TextToSpeech;
14  import android.util.Log;
15  import android.view.Menu;
16  import android.view.MenuItem;
17  import android.view.View;
18  import android.widget.Button;
19  import android.widget.TextView;
20
21  import com.hoho.android.usbserial.driver.UsbSerialDriver;
22  import com.hoho.android.usbserial.driver.UsbSerialPort;
23  import com.hoho.android.usbserial.driver.UsbSerialProber;
24  import com.hoho.android.usbserial.util.HexDump;
25  import com.hoho.android.usbserial.util.SerialInputOutputManager;
26
27  import java.io.BufferedReader;
28  import java.io.IOException;
29  import java.io.InputStreamReader;
30  import java.io.OutputStream;
31  import java.io.PrintWriter;
32  import java.net.ServerSocket;
33  import java.net.Socket;
34  import java.util.ArrayList;
35  import java.util.List;
36  import java.util.Locale;
37  import java.util.concurrent.ExecutorService;
38  import java.util.concurrent.Executors;
39
40  import course.examples.intelligentrobot.R;
41
42  public class WirelessServiceActivity extends LifecycleLoggingActivity implements
    TextToSpeech.OnInitListener {
43      private final String TAG = WirelessServiceActivity.class.getSimpleName();
44
45      //USB communication
46      private static UsbSerialPort sPort = null;
47      private SerialInputOutputManager mSerialIoManager;
48      private final ExecutorService mExecutor = Executors.newSingleThreadExecutor();
49      private UsbManager mUsbManager = null;
50      private String buffer = new String();
51
52      //General
53      private TextView t_ConnectionStatus;
54      private TextView t_serviceStatus;
55      private TextView t_phoneStatus;
56      private Handler m_ConnectionStatusTextHandler = new Handler();
57      private Handler m_serviceStatusTextHandler = new Handler();
58      private Handler m_phoneStatusTextHandler = new Handler();
```

```
59      private boolean TTSready = false;
60      private MediaPlayer mediaPlayer;
61
62      //Network robot Service
63      NsdServiceInfo serviceInfo = null;
64      ServerSocket mServerSocket = null;
65      int mLocalPort = 0;
66      NsdManager.RegistrationListener mRegistrationListener = null;
67      NsdManager mNsdManager = null;
68      Thread serviceThread = null;
69      Thread serverThread = null;
70      private boolean clientForcedClose = false;
71      private PrintWriter socketOutput = null;
72
73      //TTS fields
74      private int MY_DATA_CHECK_CODE = 1234;
75      private TextToSpeech mTts;
76
77      @Override
78      protected void onCreate(Bundle savedInstanceState) {
79          super.onCreate(savedInstanceState);
80          setContentView(R.layout.activity_drive_service);
81
82
83          t_ConnectionStatus = (TextView) findViewById(R.id.textView_robo_status);
84          t_serviceStatus = (TextView) findViewById(R.id.textView_service_status);
85          t_phoneStatus = (TextView) findViewById(R.id.textView_phoneConnection);
86
87          mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
88
89          Button button_refresh = (Button) findViewById(R.id.button_Refresh);
90          button_refresh.setOnClickListener(new View.OnClickListener() {
91              public void onClick(View v) {
92                  // serial connection
93                  refreshDeviceList();
94                  openPort();
95                  // socket and services
96                  tearDown();
97                  initializeSocketService();
98              }
99          });
100
101         Intent checkIntent = new Intent();
102         checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
103         startActivityForResult(checkIntent, MY_DATA_CHECK_CODE);
104
105
106     }
107
108     @Override
109     protected void onDestroy(){
110         super.onDestroy();
111     }
112
113     @Override
114     protected void onStart(){
115         // set up serial connection
116         refreshDeviceList();
117         openPort();
```

```
118         // setup socket and service
119         initializeSocketService();
120         super.onStart();
121     }
122
123     @Override
124     protected void onStop() {
125         // close socket and services
126         tearDown();
127         // close serial connection
128         closePort();
129         super.onStop();
130     }
131
132
133     @Override
134     public boolean onCreateOptionsMenu(Menu menu) {
135         // Inflate the menu; this adds items to the action bar if it is present.
136         //getMenuInflater().inflate(R.menu.menu_drive, menu);
137         return true;
138     }
139
140     @Override
141     public boolean onOptionsItemSelected(MenuItem item) {
142         // Handle action bar item clicks here. The action bar will
143         // automatically handle clicks on the Home/Up button, so long
144         // as you specify a parent activity in AndroidManifest.xml.
145         int id = item.getItemId();
146
147         //noinspection SimplifiableIfStatement
148 //        if (id == R.id.action_settings) {
149 //            return true;
150 //        }
151
152         return super.onOptionsItemSelected(item);
153     }
154
155     private void initializeSocketService(){
156         Log.d(TAG, "initializeSocketService()");
157         mNsdManager = (NsdManager) this.getSystemService(Context.
    NSD_SERVICE);
158         serviceThread = new Thread(new DriveServiceThread() );
159         serviceThread.start();
160     }
161
162     private void tearDown(){
163         Log.d(TAG, "tearDown()");
164         clientForcedClose = false;
165         if(this.serverThread != null) {
166             if(serverThread.isInterrupted()){
167                 // client called End of stream
168             }
169             else{
170                 this.serverThread.interrupt();
171             }
172             this.serverThread = null;
173         }
174         if(mRegistrationListener != null){
175             mNsdManager.unregisterService(mRegistrationListener);
```

```
176             mRegistrationListener = null;
177         }
178         if(mServerSocket != null){
179             try {
180                 mServerSocket.close();
181                 mServerSocket = null;
182             } catch (IOException e) {
183                 e.printStackTrace();
184             }
185         }
186         if(this.serviceThread != null) {
187             this.serviceThread.interrupt();
188             this.serviceThread = null;
189         }
190     }
191 
192     // Service methods
193     class DriveServiceThread implements Runnable {
194         @Override
195         public void run(){
196             Log.d(TAG, "DriveServiceThread run()");
197             Socket socket = null;
198             try {
199                 mServerSocket = new ServerSocket(0);
200             } catch (IOException e) {
201                 e.printStackTrace();
202                 mLocalPort = 0;
203             }
204             mLocalPort = mServerSocket.getLocalPort();
205             setClientStatusText("Client to Connect Port " + String.valueOf(mLocalPort)
    );
206             registerDriveService(mLocalPort);
207             while(!Thread.currentThread().isInterrupted()){
208                 Log.d(TAG, "DriveServiceThread run() while");
209                 try{
210                     Log.d(TAG, "Waiting for client in run() before mServerSocket.
    accept()");
211                     socket = mServerSocket.accept();
212                     Log.d(TAG, "Waiting for client in run() after mServerSocket.accept
    ()");
213                     serverThread = new Thread(new ServerCommsClientThread(socket) );
214                     serverThread.start();
215                     setClientStatusText("Client connected!");
216                 }catch (Exception e){
217                     e.printStackTrace();
218                 }
219             }
220         }
221     }
222 
223     class ServerCommsClientThread implements Runnable {
224         private Socket serverSocket;
225         private BufferedReader input;
226         public ServerCommsClientThread(Socket serverSocket) {
227             Log.d(TAG, "ServerCommsclientThread()");
228             this.serverSocket = serverSocket;
229             try {
230                 this.input = new BufferedReader(new InputStreamReader(this.
    serverSocket.getInputStream()));
```

```
231             OutputStream out = this.serverSocket.getOutputStream();
232             socketOutput = new PrintWriter(out);
233          } catch (IOException e) {
234             e.printStackTrace();
235          }
236       }
237       @Override
238       public void run() {
239          Log.d(TAG, "ServerCommsclientThread run()");
240          while (!Thread.currentThread().isInterrupted()) {
241             try {
242                if(input.ready()) {
243                   String read = input.readLine();
244                   if(read.compareTo("EOS") == 0 || read==null ){
245                      // end of stream
246                      setClientStatusText("Client disconnected!");
247                      clientForcedClose = true;
248                      Thread.currentThread().interrupt();
249                   }
250                   else if(read!=null && read.charAt(0)=='$')
251                   {
252                      //speak the phrase...
253                      speakTTS(read.substring(1));
254                   }
255                   else if(read != null) {
256                      runCommand(read);
257                      //setClientStatusText("Received: " + read);
258                   }
259                }
260             }
261             catch (Exception e){
262                e.printStackTrace();
263             }
264          }
265
266          if(!clientForcedClose){
267             //server side close
268             //server sends EOS
269             send_EOS();
270          }
271          // close socket
272          try {
273             this.serverSocket.close();
274          } catch (IOException e) {
275             e.printStackTrace();
276          }
277
278          if(clientForcedClose) {
279             //close
280             tearDown();
281             // reopen
282             initializeSocketService();
283             clientForcedClose = false;
284          }
285       }
286    }
287
288    public boolean registerDriveService(int port) {
289       Log.d(TAG, "registerDriveService()");
```

```
290         serviceInfo = new NsdServiceInfo();
291         serviceInfo.setServiceName("SmartBot");
292         serviceInfo.setServiceType("_http._udp.");
293         serviceInfo.setPort(port);
294
295         initializeRegistrationListener();
296         if (mRegistrationListener != null){
297            mNsdManager.registerService(
298                  serviceInfo, NsdManager.PROTOCOL_DNS_SD,
    mRegistrationListener);
299            return true;
300         }
301         else{
302            setServiceStatusText("Listener Failed!");
303            return false;
304         }
305      }
306
307      public void initializeRegistrationListener() {
308         Log.d(TAG, "initializeRegistrationListener()");
309         mRegistrationListener = new NsdManager.RegistrationListener() {
310
311            @Override
312            public void onServiceRegistered(NsdServiceInfo info) {
313               // Save the service name.  Android may have changed it in order to
314               // resolve a conflict, so update the name you initially requested
315               // with the name Android actually used.
316               String mServiceName = serviceInfo.getServiceName();
317               setServiceStatusText(mServiceName+ " Registered");
318               Log.d(TAG, "onServiceRegistered() registered as " + mServiceName);
319
320            }
321
322            @Override
323            public void onRegistrationFailed(NsdServiceInfo serviceInfo, int errorCode)
    {
324               Log.d(TAG, "onRegistrationFailed()");
325
326               // Registration failed!  Put debugging code here to determine why.
327               String mServiceName = serviceInfo.getServiceName();
328               setServiceStatusText(mServiceName + " Registration Failed!");
329            }
330
331            @Override
332            public void onServiceUnregistered(NsdServiceInfo arg0) {
333               Log.d(TAG, "onServiceUnregistered()");
334
335               // Service has been unregistered.  This only happens when you call
336               // NsdManager.unregisterService() and pass in this listener.
337               String mServiceName = serviceInfo.getServiceName();
338               setServiceStatusText(mServiceName + " Unregistered Successfully!");
339            }
340
341            @Override
342            public void onUnregistrationFailed(NsdServiceInfo serviceInfo, int
    errorCode) {
343               Log.d(TAG, "onUnregistrationFailed()");
344               // Unregistration failed.  Put debugging code here to determine why.
345               String mServiceName = serviceInfo.getServiceName();
```

```
346                 setServiceStatusText(mServiceName + " Unregisteration Failed!");
347             }
348         };
349     }
350
351     //   Serial Comms Methods
352     private final SerialInputOutputManager.Listener mSerial_IO_Listener =
353         new SerialInputOutputManager.Listener() {
354
355             @Override
356             public void onRunError(Exception e) {
357                 Log.d(TAG, "Runner stopped.");
358             }
359
360             @Override
361             public void onNewData(final byte[] data) {
362                 WirelessServiceActivity.this.runOnUiThread(new Runnable() {
363                     @Override
364                     public void run() {
365                         WirelessServiceActivity.this.updateReceivedData(data);
366                     }
367                 });
368             }
369         };
370
371     private void updateReceivedData(byte[] data) {
372         String newData = new String(data, 0, data.length);
373         String delimit = "\r\n";
374         buffer = buffer.concat(newData);
375         int endPos = buffer.indexOf(delimit);
376
377         if (endPos >= 0) {
378             String msg = buffer.substring(0, endPos);
379             if (msg.contains("Ack")) {
380                 //show nothing
381             } else {
382                 if(socketOutput != null){
383                     socketOutput.println(msg);
384                     socketOutput.flush();
385                 }
386             }
387             // remove read message from buffer
388             buffer = buffer.substring(endPos + delimit.length(), buffer.length());
389         }
390     }
391
392     private void onDeviceStateChange() {
393         stopIoManager();
394         startIoManager();
395     }
396
397     private void stopIoManager() {
398         if (mSerialIoManager != null) {
399             Log.i(TAG, "Stopping io manager ..");
400             mSerialIoManager.stop();
401             mSerialIoManager = null;
402         }
403     }
404
```

```
405     private void startIoManager() {
406         if (sPort != null) {
407             Log.i(TAG, "Starting io manager ..");
408             mSerialIoManager = new SerialInputOutputManager(sPort,
    mSerial_IO_Listener);
409             mExecutor.submit(mSerialIoManager);
410         }
411     }
412 
413     private void refreshDeviceList() {
414         List<UsbSerialDriver> drivers = UsbSerialProber.getDefaultProber().
    findAllDrivers(mUsbManager);
415 
416         final List<UsbSerialPort> result = new ArrayList<UsbSerialPort>();
417         for (final UsbSerialDriver driver : drivers) {
418             final List<UsbSerialPort> ports = driver.getPorts();
419             Log.d(TAG, String.format("+ %s: %s port%s",
420                     driver, Integer.valueOf(ports.size()), ports.size() == 1 ? "" : "s"));
421             result.addAll(ports);
422         }
423 
424         UsbSerialDriver driver = null;
425         UsbDevice device = null;
426 
427         if (result.size() > 0) {
428             sPort = result.get(0);
429             driver = sPort.getDriver();
430             device = driver.getDevice();
431             String title = String.format("Vendor %s Product %s",
432                     HexDump.toHexString((short) device.getVendorId()),
433                     HexDump.toHexString((short) device.getProductId()));
434             String subtitle = driver.getClass().getSimpleName();
435             setConnectionStatusText(title + subtitle);
436         } else {
437             sPort = null;
438             setConnectionStatusText("No Port Found!");
439         }
440     }
441 
442     private void openPort() {
443         if (sPort == null) {
444             setConnectionStatusText(" No serial device.");
445         } else {
446             mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
447             UsbDevice device = sPort.getDriver().getDevice();
448             UsbDeviceConnection connection = mUsbManager.openDevice(device);
449             if (connection == null) {
450                 setConnectionStatusText(" Opening device failed");
451                 return;
452             }
453             try {
454                 sPort.open(connection);
455                 sPort.setParameters(19200, 8, UsbSerialPort.STOPBITS_1, UsbSerialPort.
    PARITY_NONE);
456             } catch (IOException e) {
457                 Log.e(TAG, "Error setting up device: " + e.getMessage(), e);
458                 setConnectionStatusText("Error opening device: " + e.getMessage());
459                 try {
460                     sPort.close();
```

```
461                 } catch (IOException e2) {
462                     // Ignore.
463                 }
464                 sPort = null;
465                 return;
466             }
467             setConnectionStatusText("Serial device: " + sPort.getClass().getSimpleName
    ());
468         }
469         onDeviceStateChange();
470     }
471
472     private void closePort() {
473         stopIoManager();
474         if (sPort != null) {
475             try {
476                 sPort.close();
477             } catch (IOException e) {
478                 // Ignore.
479             }
480             sPort = null;
481         }
482     }
483
484     private void send_L(int speed) {
485         if(sPort!=null){
486             String cmd = "L";
487             cmd = cmd.concat(String.valueOf(speed)+"\r\n");
488             try {
489                 sPort.write(cmd.getBytes(), 100);
490                 Log.d(TAG, "send_L() sent left!");
491             }catch(Exception ioE){
492                 Log.e(TAG, ioE.getMessage());
493             }
494         }
495     }
496
497     private void send_R(int speed) {
498         if(sPort != null){
499             String cmd = "R";
500             cmd = cmd.concat(String.valueOf(speed)+"\r\n");
501             try {
502                 sPort.write(cmd.getBytes(), 100);
503                 Log.d(TAG, "send_R() sent right!");
504             }catch(Exception ioE){
505                 Log.e(TAG, ioE.getMessage());
506             }
507         }
508     }
509
510     private void send_C() {
511         if(sPort != null){
512             String cmd = "C" + "\r\n";
513             try {
514                 sPort.write(cmd.getBytes(), 100);
515             }catch(Exception ioE){
516                 Log.e(TAG, ioE.getMessage());
517             }
518         }
```

```
519     }
520
521     private void send_EOS(){
522         if(socketOutput != null){
523             socketOutput.println("EOS");
524             socketOutput.flush();
525         }
526     }
527
528     public void setConnectionStatusText(String txt){
529         final String msg = txt;
530         m_ConnectionStatusTextHandler.post(new Runnable() {
531             @Override
532             public void run() {
533                 t_ConnectionStatus.setText(msg);
534             }
535         });
536     }
537
538
539     public void setServiceStatusText(String txt){
540         final String msg = txt;
541         m_serviceStatusTextHandler.post(new Runnable() {
542             @Override
543             public void run() {
544                 t_serviceStatus.setText(msg);
545             }
546         });
547     }
548
549     public void setClientStatusText(String txt){
550         final String msg = txt;
551         m_phoneStatusTextHandler.post(new Runnable() {
552             @Override
553             public void run() {
554                 t_phoneStatus.setText(msg);
555             }
556         });
557     }
558
559     void runCommand(String cmd){
560         if(cmd.contains("R")){
561             String paramStr = cmd.substring(1).trim();
562             if(!paramStr.isEmpty()){
563                 int param = Integer.parseInt(paramStr);
564                 send_R(param);
565             }
566         }
567         else if(cmd.contains("L")){
568             String paramStr = cmd.substring(1).trim();
569             if(!paramStr.isEmpty()){
570                 int param = Integer.parseInt(paramStr);
571                 send_L(param);
572             }
573         }
574         else if(cmd.compareTo("C")==0){
575             send_C();
576         }
577     }
```

```
578
579     public void speakTTS(String speakString)
580     {
581         if(speakString == null)
582         {
583             speakString = "null, null";
584         }
585
586         Log.d(TAG, "speakTTS() speakstring: " + speakString);
587
588         if(mTts == null)
589         {
590             Log.d(TAG, "speakTTS() mTts null");
591         }
592
593         Log.d(TAG, "speakTTS() charAt(0) == '*': " + (speakString.charAt(0)=='*'))
    ;
594
595         if(speakString.charAt(0)=='*')
596         {
597             Log.d(TAG, "speakTTS() charAt(0) == '*'");
598             //  ImageView pic = (ImageView) findViewById(R.id.robotFace);
599             //  pic.setImageResource(R.drawable.sadpuppy);
600             puppyDance();
601
602             try{
603                 mediaPlayer = MediaPlayer.create(this, R.raw.puppy);
604                 mediaPlayer.setOnCompletionListener(new MediaPlayer.
    OnCompletionListener() {
605                     @Override
606                     public void onCompletion(MediaPlayer mp) {
607                       //maybe make a puppy thread??
608                       //and then change the picture back?
609                       //  send_L(0);
610                       //  send_R(0);
611                         Log.d(TAG, "speakTTS(), mediaPlayer ended");
612
613                     }
614                 });
615                 mediaPlayer.start();
616                 Log.d(TAG, "speakTTS(), mediaPlayer started");
617             }catch(Exception e)
618             {
619                 Log.e(TAG,""+e);
620                 Log.d(TAG, "speakTTS(), error in media player: " + e);
621             }
622
623         }
624         else if(TTSready)
625         {
626             mTts.speak(speakString, TextToSpeech.QUEUE_FLUSH, null);
627           //  speechTextView.setText("Response: " + speakString);
628         }
629     }
630
631     @Override
632     protected void onActivityResult(
633             int requestCode, int resultCode, Intent data) {
634         if (requestCode == MY_DATA_CHECK_CODE) {
```

```
635             if (resultCode == TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
636                 // success, create the TTS instance
637                 Log.d(TAG, "onActivityResult() mTts created");
638                 mTts = new TextToSpeech(this, this);
639 
640             } else {
641                 // missing data, install it
642                 Log.d(TAG,"onActivityResult() mTts failed, installed new intent");
643                 Intent installIntent = new Intent();
644                 installIntent.setAction(
645                         TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
646                 startActivity(installIntent);
647             }
648         }
649     }
650 
651     @Override
652     public void onInit(int status)
653     {
654         mTts.setLanguage(Locale.ENGLISH);
655         TTSready = true;
656     }
657 
658     public void puppyDance()
659     {
660         long timeOffset = 0;
661         m_phoneStatusTextHandler.postDelayed(new Runnable() {
662             @Override
663             public void run() {
664                 Log.d(TAG, "speakTTS() move right post delayed");
665                 //right
666                 send_L(400);
667                 send_R(-400);
668             }
669         }, timeOffset+=1000);
670 
671         m_phoneStatusTextHandler.postDelayed(new Runnable() {
672             @Override
673             public void run() {
674                 Log.d(TAG, "speakTTS() stop post delayed");
675 
676                 //left
677                 send_L(-400);
678                 send_R(400);
679             }
680         }, timeOffset+=1000);
681 
682         m_phoneStatusTextHandler.postDelayed(new Runnable() {
683             @Override
684             public void run() {
685                 Log.d(TAG, "speakTTS() stop post delayed");
686 
687                 //left
688                 send_L(400);
689                 send_R(400);
690             }
691         }, timeOffset+=1000);
692 
693         m_phoneStatusTextHandler.postDelayed(new Runnable() {
```

```
694             @Override
695             public void run() {
696                 Log.d(TAG, "speakTTS() stop post delayed");
697
698                 //left
699                 send_L(-400);
700                 send_R(-400);
701             }
702         }, timeOffset+=1000);
703
704         m_phoneStatusTextHandler.postDelayed(new Runnable() {
705             @Override
706             public void run() {
707                 Log.d(TAG, "speakTTS() stop post delayed");
708
709                 //left
710                 send_L(0);
711                 send_R(0);
712             }
713         }, timeOffset+=1000);
714     }
715
716
717
718 }
719
```