

EE 4830

Final Paper

Stephen Boyer
Electrical and Computer Engineering Department
College of Engineering
University of Wyoming
Laramie, WY 82071
sboyer3@uwyo.edu

Abstract

This project consists of interfacing an application running on a Google Android device with two external devices to implement an infrared television controller. The first of the two external devices is a *hub* device to connect to the user's existing wireless fidelity (WiFi) network. It has a radio frequency transmitter enabling it to communicate with the second external device, a battery-powered *child* device. This device contains an IR receiver and transmitter pair for learning IR codes from a television remote and sending them to a television, respectively.

This project was chosen mainly to reduce the number of devices a user must keep track of. Once the remote codes are learned by the system, a dedicated television remote is redundant. Since, unlike with infrared communication, radio frequency and WiFi communication do not require line of sight, users are not required to point anything at the television or even be in the same room. Conveniently, many people already own a phone or tablet running the Android operating system. With this system, an application could easily be developed to work with Apple's iOS or essentially any platform allowed to run third-party programs.

Keywords: Arduino, Android, Raspberry Pi, infrared, television, remote, LED, RF

Overview

This television control system was completed over the past year. From the start of the project, the goal was to make an adaptable system for controlling one or more televisions using an infrared (IR) light-emitting diode (LED). This works similarly to how a normal television remote works. Up to three televisions are supported by the Android application although enough equipment was only made for one. In order to control more televisions, a user would only need more *child* devices as the *hub* can communicate with more than one.

The system functions by sending commands over wireless fidelity (WiFi) to the *hub* device which forwards them to the *child* device over radio frequency (RF) communication. A special

feature of this system is that of the *child* device to “learn” IR remote codes. This means that any brand or model of television is supported as long as its remote uses IR.

Included in this paper are background, technical, testing, and packaging sections. These are followed by other considerations and lessons learned. Also included are a schedule, parts list, circuit diagram, construction diagram, structure and activity diagrams, and code.

Background

Communication between devices is the most important factor of this project. The first aspect of this is communication between the Android application and the *hub* device. WiFi was chosen for this as it provides good range and is already used in homes everywhere. The downside is that it uses a large amount of power which is why the *hub* device runs off of a wall adapter instead of batteries. The second aspect of the communication problem is between the *hub* and *child* devices. RF communication at a frequency of 315 megahertz (MHz) was chosen for this for its relative simplicity and low power requirements. The power requirements allow the *child* to run off of battery power. The final type of communication used is infrared between the *child* and television being controlled. Most TVs today use infrared light with a wavelength of 950 nanometers for control so this was the only option.

Google Android is a Linux-based smart phone operating system (OS) originally released in 2009. In this project, Android 4.4 or “KitKat” is used. The application will run on any newer version as well. Android applications are programmed and designed using the Android Software Development Kit (SDK) and elements of the Java Development Kit (JDK). Java is the predominant programming language used here. The Android SDK defines the base requirements an application must have. For starters, a `MainActivity.java` file is used to define and implement the main Activity for an application. In Android, Activities are used as windows in which user interaction takes place. An `AndroidManifest.xml` file is required for defining the Activities in use in the application, any permissions the application requires from the Operating System, and any Intents used. The only permission used here is for Internet access. Although the Internet is not actually accessed, WiFi access requires this permission regardless. In Android, Intents are used for opening Activities. Here, Intents are used for opening the application from the launcher in Android as well as opening an Activity where settings are displayed. A `build.gradle` file defines the version of Android an application is being compiled against. Here, SDK version 19 is used. This represents Android 4.4. According to the Android Open Source Project, 47% of Android devices use this version or above [1].

Technical Project Description

This project included three distinct parts: an Android application running on a smartphone, a *hub* device for relaying commands, and a *child* device for receiving commands and forwarding them to a television. Together, these parts make up a system for controlling a television easily.

Other than powering on the *hub* and *child* devices, user interaction is done exclusively with the Android application. Once the user presses a virtual button, a signal is sent to

the *hub* device over WiFi. The device in question is a Raspberry Pi B+ with a Universal Serial Bus (USB) WiFi adapter. It also has a small printed circuit board (PCB) with a RF transmitter attached. The *hub* responds by sending a corresponding signal to the *child* device via its RF transmitter. This signal is received by the RF receiver on the *child* device, an Arduino UNO R3. Depending on the command, the *child* turns on its IR receiver to wait for a signal from a remote control, turns on its IR LED and sends a signal to a television, or erases all codes from its memory.

Android Application

The first part of the system is the Android application running on a smartphone with Android 4.4 or above. As is typical of Android applications, this is programmed in the Java programming language. The Extensible Markup Language (XML) is utilized for defining the build configuration, interface, preferences, and some variables.

The interface of the application is a grid of buttons for the various functions of a standard remote control. In addition to buttons numbered 0-9 for channels, there are buttons for power, input source, volume up and down, channel up and down, and mute. There is also a button labeled “Favorites”, the pressing of which causes a list of favorite channels to pop up. In the upper right corner of the application is a button labeled “Settings” which takes the user to a page where preferences such as Internet Protocol (IP) address, port number, password, and username needed to communicate with the *hub* device. These settings should not have to be changed for this project, but it is a good idea to make them editable for users. The last two settings are “Reset Recorded Buttons” and “Set Favorite Channel”. “Reset Recorded Buttons” causes the app to send a signal to the *hub* to cause the *child* to erase all stored IR codes. There are three horizontal bars in the upper left corner of the application. When pressed, a menu slides out from the left and allows the user to change to other TVs or *child* devices. The final button in the interface is a switch at the bottom. When toggled by pressing, this switch changes the mode of operation of the app from “Button Mode”, for sending recorded button presses to the *hub*, to “Record Mode” for sending a signal to the *hub* representing a command to record along with the button pressed.

The commands being sent from the application are Hypertext Transfer Protocol (HTTP) POST requests. An HTTP server running on the Raspberry Pi-based *hub* device listens for these requests and acts on them. The request takes the form of a URL as follows:

`http://[IP Address of Hub]:[Server Port]/macros/[Command Type]/[Button Number]`.

JSON (JavaScript Object Notation) is used for sending requests. The password and username are sent to the *hub* using a JSONParser which is used along with the URL above to create a JSONObject. At the creation of this object, the command itself is sent out over the Android device’s WiFi connection. Because of requirements in Android, this process is completed in a thread separate from the rest of the application. This thread starts when a button is pressed and stops after a command is sent.

Hub Device

The second part of the system is the *hub*. It is based on a Raspberry Pi B+ running Raspbian OS. Like Android, this operating system is based on Linux. To create the listening server, a library called WebIOPi was used. This framework is developed by Eric Ptak and released under the Apache license [2]. WebIOPi is developed in Java with a Python-based Application Program Interface (API). It simplifies the creation of an HTTP server capable of listening for the POST from another device. In this case, it listens for commands from the Android app. The WebIOPi library is used through the running of a Python script (see Listing 9 in the Code section of the Appendix) that starts a server on the Raspberry Pi's IP address with a port number defined in a configuration file located on its Secure Digital (SD) card at `/etc/config/webiopi`. The default is port 8000. In this same file, the username and password for the server can be modified. Naturally, to function correctly, the settings configured here must match those in the settings section of the Android application.

When a command is received, the corresponding function is called. For instance, a command of the form

`http://10.10.0.1:8000/macros/sendButton/1`

will be received by the hub at 10.10.0.1 via port 8000. The `sendButton(button)` function in the Python script will be called with an argument 1 for `button`. The function `sendButton` blinks the yellow status LED once before sending a shortened representative command to the *child* via RF. The shortened command takes the form

`[Child Number][Command Type][Button Number]`.

In the above example, Child Number would be 0 for the one and only *child* created in the scope of this project. This field is only present so that it is possible to add more *child* devices if the user desires them. If this number does not match the number of the *child* receiving it, the *child* will ignore the command. For command type, there are three possible values: 0 for `sendButton`, 1 for `sendRecord`, and 2 for `resetChild`. `Button Number` is simply an integer representing the number the button is assigned in the Android application. The numbers are only used for keeping track of what button was pressed and are meaningless otherwise. For the current application with 17 total buttons, the numbers range from 0 to 16.

The final part of the *hub*'s PCB is an antenna connected to the antenna pin of the RF transmitter. It is a simple piece of wire standing vertically. Its length is approximately 23.8cm, which is equivalent to a quarter of the wavelength of a 315 MHz signal. This is according to the formula for wavelength,

$$\gamma = \frac{v}{f}$$

[3], where γ is wavelentch, v is velocity, and f is frequency. In this case, $v = 299,792,458 \frac{\text{meters}}{\text{second}}$, or the speed of light. For a frequency of 315 MHz, $f = 315,000,000$. One quarter of this value is found to be 23.8cm.

Child Device

The final device is the one responsible for receiving commands from the *hub* and is known as the *child*. It is based on an Arduino Uno R3, surface-mount device (SMD) edition. In this project's original description, the *child* was to be based on a Texas Instruments MSP430. However, due to the small Random Access Memory (RAM) size of this device, this was deemed impossible. Texas Instruments does not make a device in this line with more RAM. The Arduino Uno R3's 2 kilobyte RAM size is four times that of the MSP430 and is more than adequate.

The *child* has three roles: recording IR remote codes, storing them in its non-volatile Electrically Erasable Programmable Read-Only Memory (EEPROM), and playing them back when necessary. It has an IR receiver module for recording messages and an IR LED for playing them back. There are two other LEDs, one red and one yellow. Finally, there is a 315 MHz RF receiver. The *child* and components are powered together using six AA batteries in a holder. This provides the approximately nine volts required by the Arduino. Voltage is stepped down to five volts for the remaining components with two voltage regulators. Everything is turned on and off with a slider switch mounted on the PCB with all of the other components external to the Arduino.

Receiving codes from the *hub* is done via the RF receiver. A library known as VirtualWire [4] is used to help with this. It runs on the *hub* as well as the *child*. As stated previously, these commands are received in the form

[Child Number][Command Type][Button Number].

Whereas these codes are sent from the *hub* as standard American Standard Code for Information Interchange (ASCII) characters, they are received by the *child* as hexadecimal representations of those ASCII characters. For instance, the code "0" for the Child Number translates to 0x30 in hexadecimal. This value is compared to a constant `CHILD_ID` to determine if the code is meant for the *child* receiving it. If not, the *child* ignores the command. However, if the code does match, the next step is to determine the command. This is done by comparing the second character in the command to 0 (0x30) for `sendButton`, 1 (0x31) for `sendRecord`, and 2 (0x32) for `sendReset`. Finally, if `Button Number` contains two digits, the third and fourth characters are combined to form the number used by the *child*. This is needed because the characters are sent serially. Otherwise, the third character is used by itself.

When the command is `sendRecord`, the IR receiver is turned on. It is a Vishay Electronics TSOP38238, which is designed to demodulate 38 kilohertz (kHz) IR signals. The fact that it demodulates signals means that it can be connected directly to a microcontroller. Signals leaving the IR receiver enter a digital input pin on the Arduino. Because of the demodulation, the codes come in as digital pulses. From here, a library is used to simplify recording IR remote codes. It also simplifies playing them back later on. This library is known as `IRremote` and is written by Ken Sheriff [5]. The library also helps in storing IR codes. In this case, EEPROM is used to store codes as it is non-volatile and good for long-term storage. After being processed using the library, the codes are represented by alternating marks and spaces which are equivalent to time high and time low, respectively. These times are measured in microseconds.

For the command `sendButton`, the IR LED is turned on. This LED has a wavelength of 950 nanometers just like television remotes. Using pulse-width modulation (PWM), the LED turns on and off for the time for the number of microseconds defined by marks and spaces, respectively. Humans cannot see infrared light due to its long wavelength. Any light with a wavelength above approximately 700 nanometers is invisible to humans [6]. As such, there is no way to see the 950 nanometer IR LED light up with the naked eye. To counteract this problem, a red LED runs in parallel with the IR LED to give visible feedback. Both of these LEDs utilize a 2N3904 n-channel biased junction transistor in a common-emitter amplifier arrangement to provide more current and therefore light than the Arduino Uno could manage otherwise. They are connected one of the Arduino’s PWM pins.

The final command that can be received is `sendReset`. This causes the Arduino to erase all stored IR codes. It is only sent when “Reset Recorded Buttons” is pressed in the “Settings” screen of the Android app.

A yellow LED is used to show the status of the child. It is connected to one of the Arduino’s digital out pins. This LED is turned on once the Arduino has finished booting. It flashes a certain number of times when each type of command is received; once for `sendRecord`, twice for `sendButton`, three times for `sendReset`, and four when it does not recognize a command. After blinking, the LED remains turned on steadily to show that the *child* is turned on.

The original specifications for this project included a battery life of one month for the *child*. Of course, these specifications were for a *child* built using a TI MSP430. The MSP430 was originally chosen for its low power consumption. An Arduino does not share in this quality which led to a resulting battery life of around 6 days instead of 30.

Testing

This system was tested by using it as a normal user would. To begin, a user would install the application on his/her Android device and open it. Then he/she would open the settings section of the application and adjust the “Username”, “Password”, “IP Address”, and “Port Number” sections for his/her *hub* device. The default settings in the application would usually be adequate.

The next step would be to begin “teaching” the hub device remote codes. This is done by toggling the “Record Mode” switch on the bottom of the main application screen to the “On” position. Then, the user would press a button (e.g. “Power”) on the screen and wait for the yellow status LED to light up on the “child” device. This signals that the device is waiting for an input on the infrared sensor representing a remote code. The user would press the button corresponding to the button pressed in the application (e.g. “Power”) on the TV’s remote control while pointing it toward the *child*. Then, the status LED would turn off, signifying a recorded remote code. This “teaching” process would be repeated with different buttons of the remote application until the user is satisfied.

The final step of testing would be to ensure the recorded remote codes work as expected. This would be done by toggling the “Record Mode” switch on the bottom of the main application screen to the “Off” position for playback of codes. Then, several buttons would need to be pressed to see if the TV performed the correct functions. If this test is passed,

the system is working as specified. Testing performed on April 21, 2015 confirmed this was the case.

Other tests included keeping the *hub* and *child* turned on for five days continuously to simulate real-world operating conditions. The same test as above was run to ensure everything continued to function. This test was passed as well.

Packaging

Packaging was taken care of using two clear, plastic boxes purchased from Amazon. Plastic was used instead of aluminum or another metal so that RF signals would be uninhibited. The plastic is clear for the *child* so that IR signals will travel through and so that the user can see the red and yellow status LEDs. The plastic on the *hub* is clear to match the *child*. In total, this packaging costed \$18.92. The container for the *hub* device is a small box designed for a Raspberry Pi. A small board with the red status LED and RF transmitter sits glued atop. For the *child*, a box large enough to hold the Arduino, PCB, and battery pack was used.

Other Considerations

Project Cost

The total cost of this project turned out to be \$135.43. See the table under Parts List in the Appendix for a breakdown of parts and their prices. A wireless router is not included. Originally, this was under the assumption that most people in the United States already own wireless routers. As it turns out, the Raspberry Pi can also be configured to run as a wireless router. This option was used when testing and demonstrating. The price of the Android device was not included either. Similar to a wireless router, this was done under the assumption that most people in United States have smartphones. Not all of them run Android, but those running iOS or Windows Phone would only need an application to be made for them. A television was not included either for the same reasons. Therefore, for these reasons and in order to save approximately \$500, a wireless router, smartphone, and television were not included in the price.

Ethical and Social Considerations

There are two potential ethical and social considerations with a device like this. The first comes with any Internet-enabled device: security. Knowingly designing a product that allows users to get hacked would be unethical indeed. Fortunately, this problem solved itself since there is little need to be able to control a television from outside of one's home. This means that the *hub* need not access the entire Internet. The Android application does for reasons beyond this project, but it has the benefit of the security built in to Android and into Linux itself. The second concern comes from using RF communication in an unlicensed way. If this project was to become a product, it would need to be approved by the Federal Communication Commission to use RF devices like it is. The RF devices themselves would need to be approved as well which would lead to more expensive devices.

Safety

This project has a high level of safety. For most parts, Direct Current (DC) signals are used instead of more dangerous Alternating Current (AC) signals. Light emitted by LEDs, both infrared and visible, is safe for human eyes. RF signals, WiFi included, have been deemed safe by the World Health Organization [7]:

Following a standard health risk assessment process, the Task Group concluded that there are no substantive health issues related to [Electromagnetic Fields] at levels generally encountered by members of the public.

The biggest concern stems from the batteries powering the *child* device. As with all batteries, there is a small chance of leakage and explosion. Under normal use of the system here, this should never happen.

Environmental

The environmental effects of this project are small. Like all electronics, the parts used here should be disposed of properly. The batteries used to power the *child* are the most critical part of this. According to Duracell [8], the batteries used here do not include mercury but should still be recycled.

Aesthetics

Although not as important as function for a project like this, the form one takes is not something to be overlooked. In this case, the resulting devices should be aesthetically pleasing enough that users would not mind having them in their houses. For both the *hub* and *child*, clear plastic boxes were used to house the parts and keep everything together. The boxes are plastic to prevent RF interference. They are clear so that they match and so that status LEDs can be viewed from outside.

Lessons Learned

Several lessons were learned during this project. Parts should always be ordered early enough to arrive on time with the cheapest shipping option. Shipping can get exceedingly expensive when one cannot wait very long for them to arrive. Extensive research needs to be done before ordering anything to make sure that everything will work together, and not only in theory. Programs should be mostly written before ordering hardware in order to make sure they will even fit on the hardware. The largest single problem in this project was ordering parts that were advertised to work together as expected, but which were too difficult or impossible to get working in practice.

With what is known today, some things would be changed on this project. More time would be spent researching to find better devices on which WiFi communication could be implemented on just a microcontroller. These devices would hopefully be smaller and require less power than a Raspberry Pi. They would also be cheaper and more suited for their task. Further, RF communication turned out to be difficult to implement. Parts were unreliable

and came well short of their advertised range. To get more range, more powerful parts could be used. Of course, this would use more electricity and would make a battery-powered *child* less feasible. Therefore, WiFi, which is more reliable and has greater range, would be used in all devices. These devices would come together to work as a more generalized home control system. Lighting control, power control, and even a television control could be built in this way. An Android application would still be used.

References

- [1] “Android Open Source Project.” *Developers*. Google, 6 Apr. 2015. Web. 19 Apr. 2015.
- [2] Ptak, Eric. “Webiopi - Internet of Things Framework.” *Google Code*. Google, 20 Dec. 2014. Web. 20 Apr. 2015.
- [3] “The Wave Equation.” *The Wave Equation*. The Physics Classroom, n.d. Web. 23 Apr. 2015. <http://www.physicsclassroom.com/class/waves/Lesson-2/The-Wave-Equation>.
- [4] “VirtualWire: VirtualWire Library for Arduino and Other Boards.” *VirtualWire*. Airspayce, 7 Feb. 2012. Web. 23 Apr. 2015. <http://www.airspayce.com/mikem/arduino/VirtualWire/index.html>.
- [5] Shirriff, Ken. “Ken Shirriff’s Blog.” *A Multi-Protocol Infrared Remote Library for the Arduino*. N.p., 1 Sept. 2009. Web. 20 Apr. 2015.
- [6] “What Wavelength Goes With a Color?” *What Wavelength Goes With a Color?* NASA, 15 Nov. 2011. Web. 20 Apr. 2015.
- [7] “Electromagnetic Fields and Public Health.” *WHO*. World Health Organization, June 2007. Web. 21 Apr. 2015.
- [8] “REGION.” *FAQ*. Duracell, 2014. Web. 21 Apr. 2015.
- [9] “Enclosures, Boxes, & Cases 5.9 X 3.2 X 1.8 Clear.” Amazon.com: Industrial & Scientific. *Amazon*, n.d. Web. 23 Apr. 2015. http://www.amazon.com/gp/product/B00HKJ2NIG/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1.
- [10] “Kotek Case Enclosure.” Amazon.com. *Amazon*, n.d. Web. 23 Apr. 2015. http://www.amazon.com/gp/product/B000FQIH50/ref=oh_aui_search_detailpage?ie=UTF8&psc=1.

Appendices

Schedule



		Task Name ▼	Finish ▼
1		Parts List Finalized	Sat 1/31/15
2		App and system working	Sun 3/15/15
3		Parts Ordered	Sun 3/15/15
4		Messaging system working	Mon 3/23/15
5		Android app finalized	Thu 4/23/15
6		Proof of concept	Mon 4/6/15
7		Schematic finalized	Mon 4/13/15
8		Working prototype	Thu 4/23/15
9		Final report written	Thu 4/23/15
			

Figure 1: Final timeline

Parts List

Part	Source	Unit Cost	Number	Total
Raspberry Pi	Newark.com	\$40.00	1	\$40.00
Arduino Uno	Sparkfun.com	\$29.95	1	\$29.95
Child Case	Amazon.com	\$13.71	1	\$13.71
USB Wifi Adapter	Amazon.com	\$8.99	1	\$8.99
Hub Case	Amazon.com	\$8.99	1	\$8.99
Power Cord	Amazon.com	\$5.99	1	\$5.99
Battery Case	Amazon.com	\$5.70	1	\$5.70
RF Rec	Sparkfun.com	\$4.95	1	\$4.95
RF Trans	Sparkfun.com	\$3.95	1	\$3.95
AA Battery	Amazon.com	\$0.43	6	\$2.59
IR Rec	Sparkfun.com	\$1.95	1	\$1.95
Voltage Regulator	Sparkfun.com	\$0.95	2	\$1.90
Battery Clip	Amazon.com	\$1.58	1	\$1.58
IR LED	Sparkfun.com	\$0.95	1	\$0.95
2N3904	Mouser.com	\$0.41	2	\$0.82
Switch	Amazon.com	\$0.75	1	\$0.75
Red LED	Sparkfun.com	\$0.35	2	\$0.70
Screw terminal	Amazon.com	\$0.21	3	\$0.62
330 Ω resistor	Mouser.com	\$0.10	5	\$0.50
Yellow LED	Sparkfun.com	\$0.35	1	\$0.35
1M Ω resistor	Radioshack	\$0.25	1	\$0.25
0.1 uF cap	Sparkfun.com	\$0.25	1	\$0.25
Total				\$135.43

Data Sheets

- RF Transmitter: http://cdn.sparkfun.com/datasheets/Wireless/General/TWS-BS-6_315MHz_ASK_RF_Transmitter_Module_Data_Sheet.pdf
- RF Receiver: <http://www.dipmicro.com/store/RF315PAIR>
- IR LED: <http://cdn.sparkfun.com/datasheets/Components/LED/YSL-R531FR1C-F1.pdf>
- Colored LEDs: <http://www.sparkfun.com/datasheets/Components/LED/COM-09590-YSL-R531R3.pdf>
- Voltage Regulators: <http://cdn.sparkfun.com/datasheets/Components/General/T0-220.pdf>
- 2N3904 BJT: <http://www.sparkfun.com/datasheets/Components/2N3904.pdf>

Circuit Diagrams

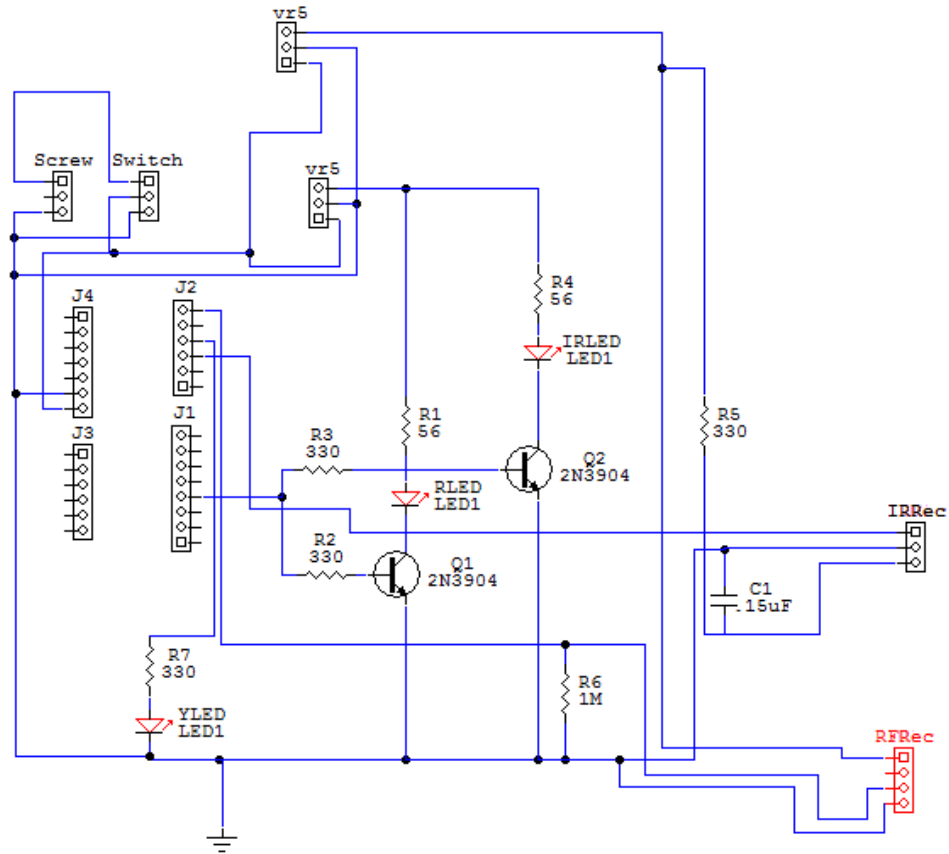


Figure 2: The circuit diagram for the *child*.

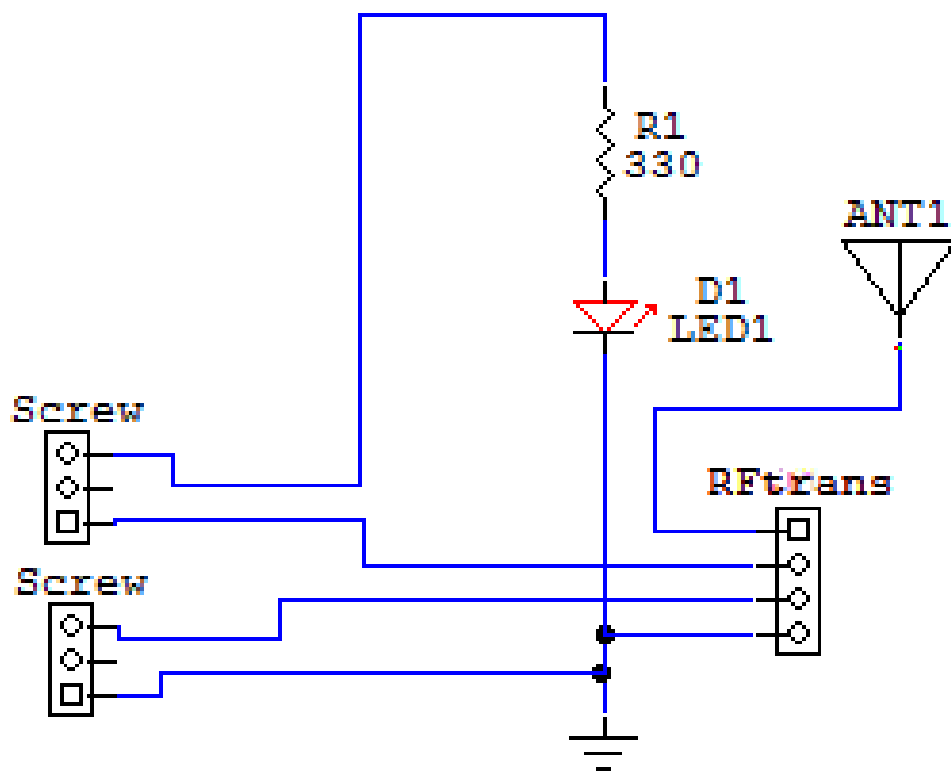


Figure 3: The circuit diagram for the *hub*.

Construction Diagrams



Figure 4: Child device packaging. [9]



Figure 5: Hub device packaging. [10]

Structure Charts for Code

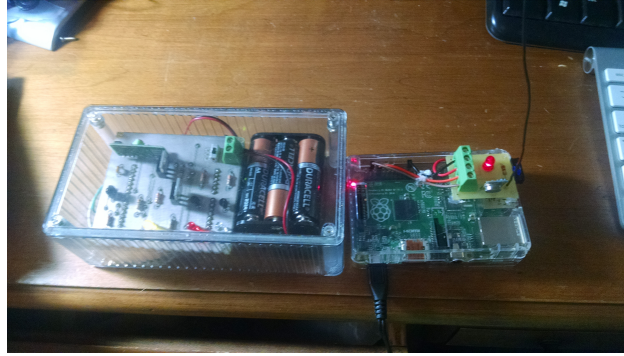


Figure 6: Final packaging.

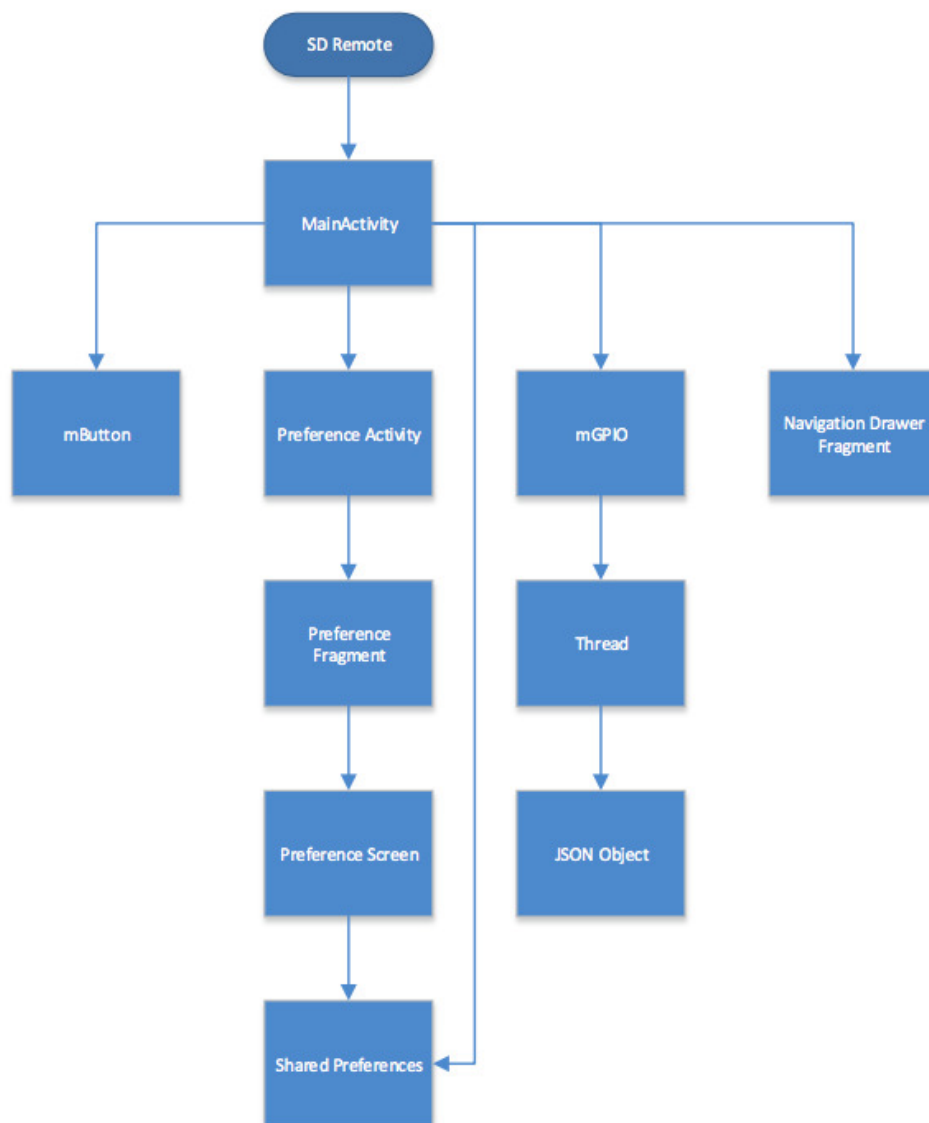


Figure 7: The Structure Chart for the Android application.

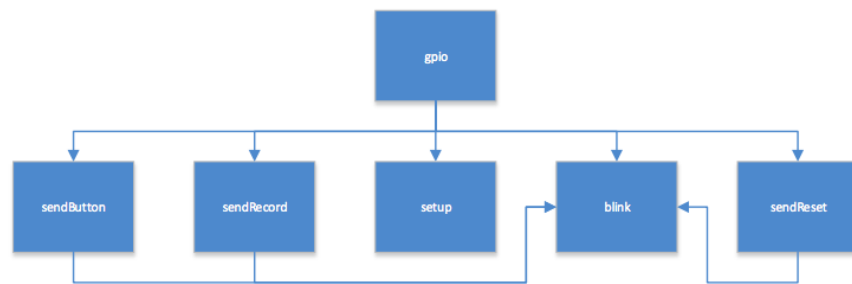


Figure 8: The Structure Chart for `gpioV2.py`, which runs on the *hub*.

UML Activity Diagrams for Code

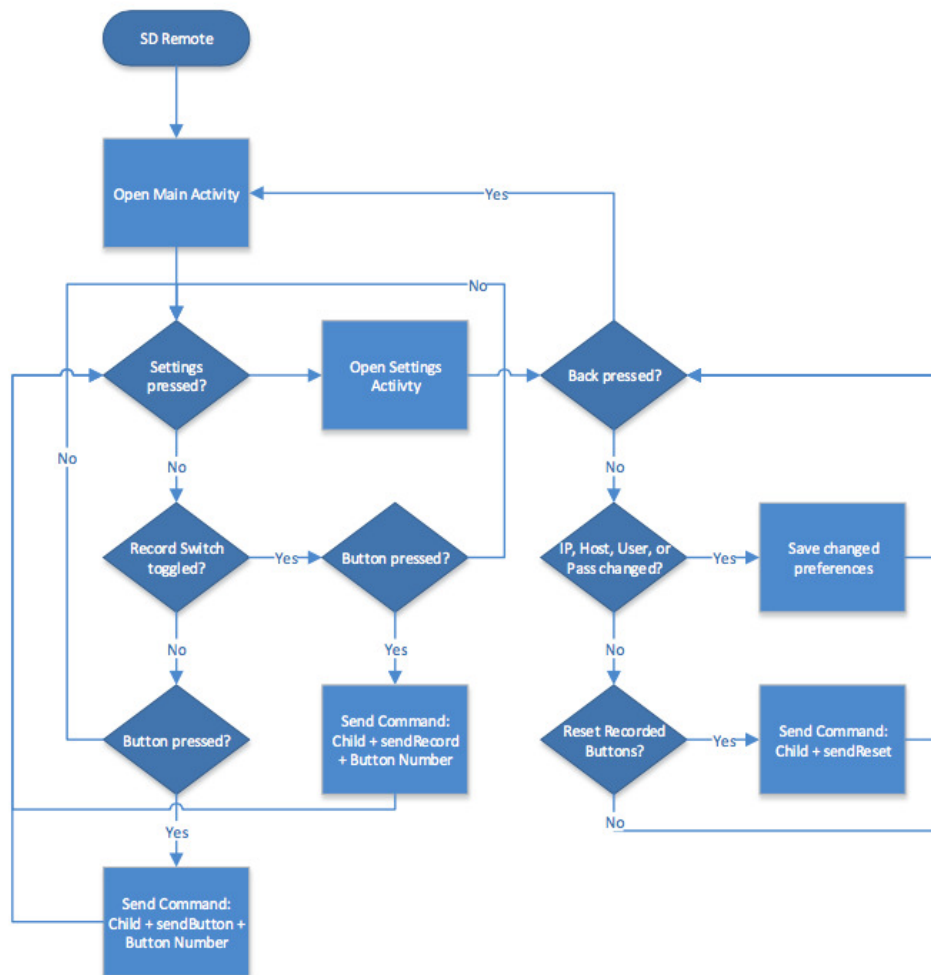


Figure 9: The Activity Diagram for the Android application.

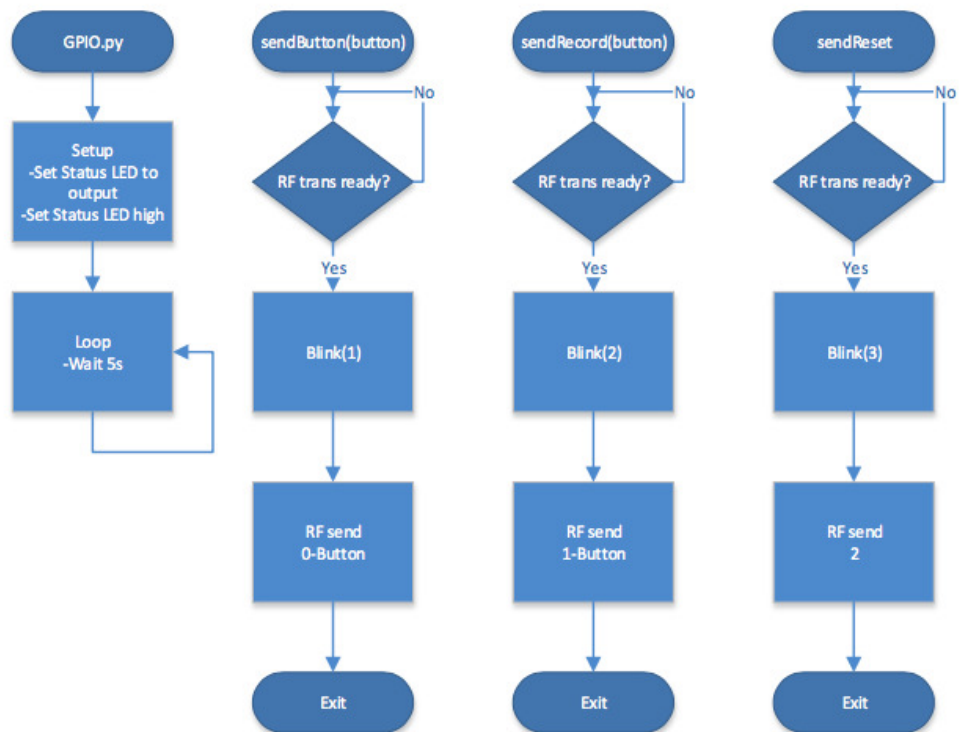


Figure 10: The Activity Diagram for `gpioV2.py` on the *hub*.

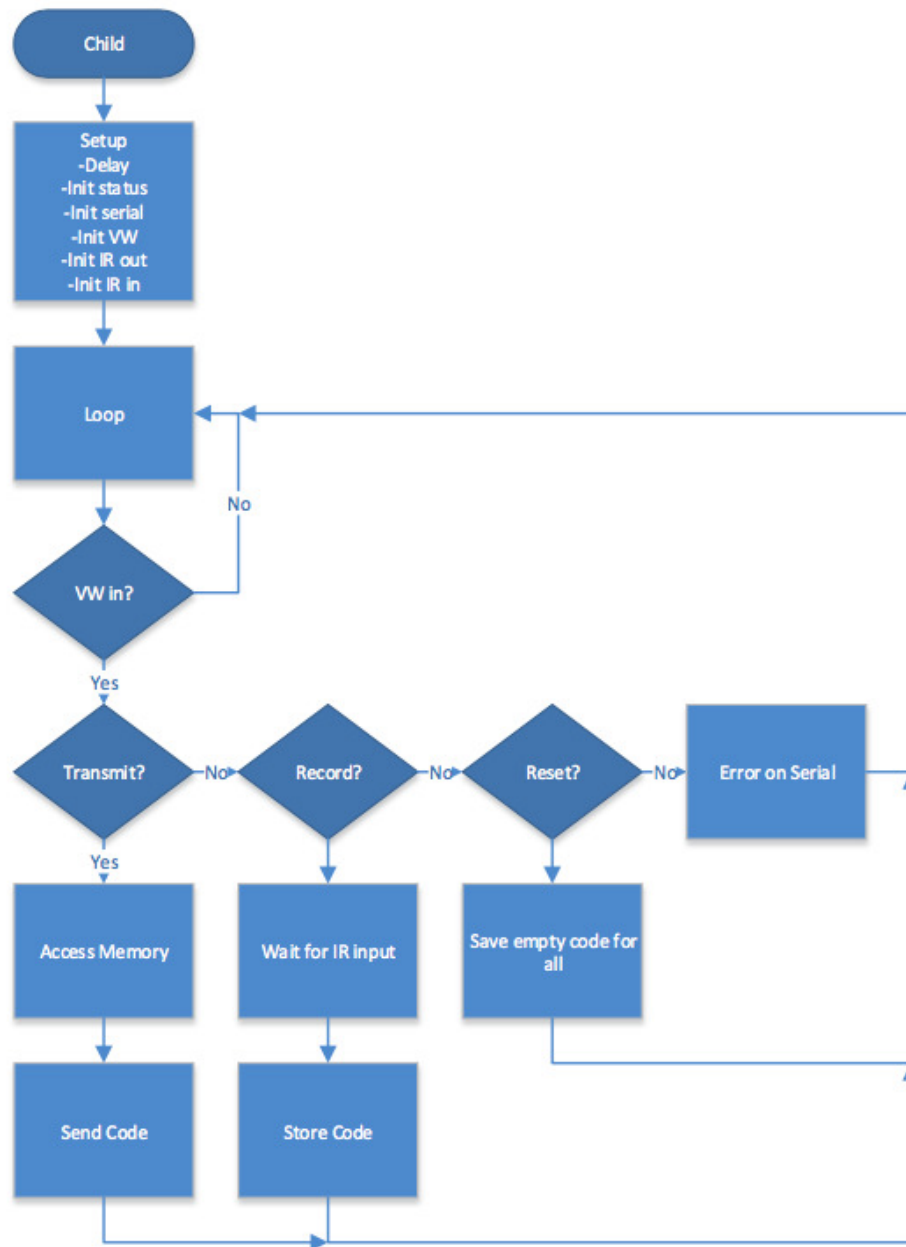


Figure 11: The Activity Diagram for `childduino.ini` on the *child*.

Code

Android Application

Listing 1: `AndroidManifest.xml`. This file is for the `SDRemote` application. It configures the application as required by Android.

```
1 <?xml version="1.0" encoding="utf-8"?>
  <manifest xmlns:android="http://schemas.android.com/apk/res/
    [+]android"
3     package="us.steveboyer.sdremote" >

5     <uses-permission
        android:name="android.permission.INTERNET"/>

7     <application
9         android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
11        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
13        <activity
            android:name=".MainActivity"
15            android:label="@string/app_name" >
            <intent-filter>
17                <action android:name="android.intent.action.MAIN"
                    [+]>/>

19                <category android:name="android.intent.category.
                    [+]LAUNCHER" />
            </intent-filter>
21        </activity>
        <activity
23            android:name=".UserSettingsActivity"
            android:label="@string/app_name" >
25            <intent-filter>
                <action android:name="android.intent.action.MAIN"
                    [+]>/>
27            </intent-filter>
            </activity>
29        </application>
    </manifest>
```

Listing 2: `MainActivity.java`. This class, which is equivalent to a `main.cpp`, contains most of the code for the Android application.

```
package us.steveboyer.sdremote;
```

2

```

import android.app.Activity;
4
import android.app.ActionBar;
6 import android.app.Fragment;
import android.app.FragmentManager;
8 import android.content.Intent;
import android.content.SharedPreferences;
10 import android.os.Bundle;
import android.preference.PreferenceManager;
12 import android.util.Log;
import android.view.LayoutInflater;
14 import android.view.Menu;
import android.view.MenuItem;
16 import android.view.View;
import android.view.ViewGroup;
18 import android.support.v4.widget.DrawerLayout;
import android.widget.PopupMenu;
20 import android.widget.Switch;
import android.widget.CompoundButton;
22 import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.Toast;
24
public class MainActivity extends Activity
26     implements NavigationDrawerFragment.
        [+]NavigationDrawerCallbacks, View.OnClickListener,
        [+]PopupMenu.OnMenuItemClickListener
    {
28     /**
        * Fragment managing the behaviors, interactions and
        [+]presentation of the navigation drawer.
30     */
    private NavigationDrawerFragment mNavigationDrawerFragment;
32
    /**
34     * Used to store the last screen title. For use in {@link #restoreActionBar\(\)}.
        */
36     private CharSequence mTitle;

    private final static int ONE = 1;
    private final static int TWO = 2;
40     private final static int THREE = 3;

    private final static int BUTTON_ZERO = 0;
    private final static int BUTTON_NULL = 1;

```

```

44     private final static int BUTTON_ONE = 17;
        private final static int BUTTON_TWO = 2;
46     private final static int BUTTON_THREE = 3;
        private final static int BUTTON_FOUR = 4;
48     private final static int BUTTON_FIVE = 5;
        private final static int BUTTON_SIX = 6;
50     private final static int BUTTON_SEVEN = 7;
        private final static int BUTTON_EIGHT = 8;
52     private final static int BUTTON_NINE = 9;
        private final static int BUTTON_CHANP = 10;
54     private final static int BUTTON_CHANM = 11;
        private final static int BUTTON_VOLP = 12;
56     private final static int BUTTON_VOLM = 13;
        private final static int BUTTON_POWER = 14;
58     private final static int BUTTON_SOURCE = 15;
        private final static int BUTTON_MUTE = 16;
60     private final static int NUM_BUTTONS = 18;


62     private SharedPreferences mainPreferences;
        private SharedPreferences.Editor prefsEditor;

64

        private PopupMenu popupMenu;
66     private mGPIO gpio;
        private Switch recordSwitch;
68     private boolean recordSwitchBool = false;
        private boolean[] buttonsEnabled;

70

    @Override
72     protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
74         setContentView(R.layout.activity_main);


76         buttonsEnabled = new boolean[NUM_BUTTONS];


78         mNavigationDrawerFragment = (NavigationDrawerFragment)
            getFragmentManager().findFragmentById(R.id.
                [+]navigation_drawer);
80         mTitle = getTitle();


82         // Set up the drawer.
        mNavigationDrawerFragment.setUp(
84             R.id.navigation_drawer,
            (DrawerLayout) findViewById(R.id.drawer_layout));

86         popupMenu = new PopupMenu(this, findViewById(R.id.

```



```

        [+]button_favs));
88 popupMenu.getMenu().add(Menu.NONE, ONE, Menu.NONE, "Item_1
        [+]");
        popupMenu.getMenu().add(Menu.NONE, TWO, Menu.NONE, "Item_2
        [+]");
90 popupMenu.getMenu().add(Menu.NONE, THREE, Menu.NONE, "Item
        [+]_3");
        popupMenu.setOnMenuItemClickListener(this);
92 findViewById(R.id.button_favs).setOnClickListener(this);

94 PreferenceManager.setDefaultValues(getApplicationContext()
        [+], R.xml.preferences, false);

96 mainPreferences = PreferenceManager.
        [+]getDefaultSharedPreferences(getApplicationContext());
        final SharedPreferences.Editor editor = mainPreferences.
        [+]edit();

98 SharedPreferences SP = PreferenceManager.
        [+]getDefaultSharedPreferences(getApplicationContext());
100 String strUsername = SP.getString("pref_user", "webiopi");
        String strPassword = SP.getString("pref_pass", "raspberry"
        [+]");
102 String strPort = SP.getString("pref_port", "8000");
        String strIP = SP.getString("pref_ip", "192.168.1.1");
104

        Log.d("Conn", "attempting_connection_on_" + strIP + "_with
        [+]_port_" + strPort + ",_pass:" + strPassword + ",_
        [+]user:" + strUsername);
106

        this.gpio = new mGPIO(new mGPIO.mConnection(strIP, Integer
        [+] .parseInt(strPort), strUsername, strPassword));
108

        mainPreferences.registerOnSharedPreferenceChangeListener(
        [+]new SharedPreferences.
        [+]OnSharedPreferenceChangeListener() {
110             public void onSharedPreferenceChanged(
                    [+]SharedPreferences prefs, String key) {
                            if (key.equalsIgnoreCase(getString(R.string.
                            [+]key_pref_reset_buttons))) {
112                                 if (mainPreferences.getBoolean(key, false)) {
                                        editor.putBoolean(getString(R.string.
                                        [+]key_pref_reset_buttons), false);
114                                         setButtonsProgrammed(false, editor);
                                                sendReset();

```

```

116         if(!recordSwitchBool) {
117             setButtonsEnabled(false);
118         }
119     }
120 }
121 }
122 });

124 prefsEditor = mainPreferences.edit();

126 recordSwitch = (Switch)findViewById(R.id.switch_record);
recordSwitch.setOnCheckedChangeListener(new
    [+]OnCheckedChangeListener() {
128     public void onCheckedChanged(CompoundButton buttonView
        [ +], boolean isChecked) {
        recordSwitchBool = isChecked;
130         if(isChecked){
            setButtonsEnabled(true);
132         } else {
            buttonsEnabled[BUTTON_ZERO] = mainPreferences.
                [ +]getBoolean("button_0_programmed", false);
134            buttonsEnabled[BUTTON_ONE] = mainPreferences.
                [ +]getBoolean("button_1_programmed", false);
            buttonsEnabled[BUTTON_TWO] = mainPreferences.
                [ +]getBoolean("button_2_programmed", false);
136            buttonsEnabled[BUTTON_THREE] = mainPreferences
                [ +].getBoolean("button_3_programmed", false)
                [ +];
            buttonsEnabled[BUTTON_FOUR] = mainPreferences.
                [ +]getBoolean("button_4_programmed", false);
138            buttonsEnabled[BUTTON_FIVE] = mainPreferences.
                [ +]getBoolean("button_5_programmed", false);
            buttonsEnabled[BUTTON_SIX] = mainPreferences.
                [ +]getBoolean("button_6_programmed", false);
140            buttonsEnabled[BUTTON_SEVEN] = mainPreferences
                [ +].getBoolean("button_7_programmed", false)
                [ +];
            buttonsEnabled[BUTTON_EIGHT] = mainPreferences
                [ +].getBoolean("button_8_programmed", false)
                [ +];
142            buttonsEnabled[BUTTON_NINE] = mainPreferences.
                [ +]getBoolean("button_9_programmed", false);
            buttonsEnabled[BUTTON_CHANP] = mainPreferences
                [ +].getBoolean("button_chanp_programmed",
                [ +]false);

```

```

144         buttonsEnabled[BUTTON_CHANM] = mainPreferences
            [+] .getBoolean("button_chanm_programmed",
                [+] false);
        buttonsEnabled[BUTTON_POWER] = mainPreferences
            [+] .getBoolean("button_power_programmed",
                [+] false);
146         buttonsEnabled[BUTTON_VOLP] = mainPreferences.
            [+]getBoolean("button_volp_programmed",
                [+] false);
        buttonsEnabled[BUTTON_VOLM] = mainPreferences.
            [+]getBoolean("button_volm_programmed",
                [+] false);
148         buttonsEnabled[BUTTON_MUTE] = mainPreferences.
            [+]getBoolean("button_mute_programmed",
                [+] false);
        buttonsEnabled[BUTTON_SOURCE] =
            [+]mainPreferences.getBoolean("
                [+]button_source_programmed", false);
150
        ((mButton)findViewById(R.id.button0)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_ZERO]);
152         ((mButton)findViewById(R.id.button1)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_ONE]);
        ((mButton)findViewById(R.id.button2)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_TWO]);
154         ((mButton)findViewById(R.id.button3)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_THREE])
            [+]];
        ((mButton)findViewById(R.id.button4)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_FOUR]);
156         ((mButton)findViewById(R.id.button5)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_FIVE]);
        ((mButton)findViewById(R.id.button6)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_SIX]);
158         ((mButton)findViewById(R.id.button7)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_SEVEN])
            [+]];
        ((mButton)findViewById(R.id.button8)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_EIGHT])
            [+]];
160         ((mButton)findViewById(R.id.button9)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_NINE]);
        ((mButton)findViewById(R.id.button_chanp)).
            [+]mSetEnabled(buttonsEnabled[BUTTON_CHANP])
            [+]];

```

```

162         ((mButton)findViewById(R.id.button_chanm)).
            [+]msetEnabled(buttonsEnabled[BUTTON_CHANM])
            [+];
        ((mButton)findViewById(R.id.button_power)).
            [+]msetEnabled(buttonsEnabled[BUTTON_POWER])
            [+];
164        ((mButton)findViewById(R.id.button_volp)).
            [+]msetEnabled(buttonsEnabled[BUTTON_VOLP]);
        ((mButton)findViewById(R.id.button_volm)).
            [+]msetEnabled(buttonsEnabled[BUTTON_VOLM]);
166        ((mButton)findViewById(R.id.button_source)).
            [+]msetEnabled(buttonsEnabled[BUTTON_SOURCE
            [+]]);
        ((mButton)findViewById(R.id.button_mute)).
            [+]msetEnabled(buttonsEnabled[BUTTON_MUTE]);

168    }
170 }
    });
172
    buttonsEnabled[BUTTON_ZERO] = mainPreferences.getBoolean("
        [+]button_0_programmed", false);
174    buttonsEnabled[BUTTON_ONE] = mainPreferences.getBoolean("
        [+]button_1_programmed", false);
    buttonsEnabled[BUTTON_TWO] = mainPreferences.getBoolean("
        [+]button_2_programmed", false);
176    buttonsEnabled[BUTTON_THREE] = mainPreferences.getBoolean(
        [+] "button_3_programmed", false);
    buttonsEnabled[BUTTON_FOUR] = mainPreferences.getBoolean("
        [+]button_4_programmed", false);
178    buttonsEnabled[BUTTON_FIVE] = mainPreferences.getBoolean("
        [+]button_5_programmed", false);
    buttonsEnabled[BUTTON_SIX] = mainPreferences.getBoolean("
        [+]button_6_programmed", false);
180    buttonsEnabled[BUTTON_SEVEN] = mainPreferences.getBoolean(
        [+] "button_7_programmed", false);
    buttonsEnabled[BUTTON_EIGHT] = mainPreferences.getBoolean(
        [+] "button_8_programmed", false);
182    buttonsEnabled[BUTTON_NINE] = mainPreferences.getBoolean("
        [+]button_9_programmed", false);
    buttonsEnabled[BUTTON_CHANP] = mainPreferences.getBoolean(
        [+] "button_chanp_programmed", false);
184    buttonsEnabled[BUTTON_CHANM] = mainPreferences.getBoolean(
        [+] "button_chanm_programmed", false);
    buttonsEnabled[BUTTON_POWER] = mainPreferences.getBoolean(

```

```

186         [+] "button_power_programmed", false);
        buttonsEnabled[BUTTON_VOLP] = mainPreferences.getBoolean("
            [+] button_volp_programmed", false);
        buttonsEnabled[BUTTON_VOLM] = mainPreferences.getBoolean("
            [+] button_volm_programmed", false);
188        buttonsEnabled[BUTTON_MUTE] = mainPreferences.getBoolean("
            [+] button_mute_programmed", false);
        buttonsEnabled[BUTTON_SOURCE] = mainPreferences.getBoolean
            [+] ("button_source_programmed", false);

190
        ((mButton)findViewById(R.id.button0)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_ZERO]);
192        ((mButton)findViewById(R.id.button1)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_ONE]);
        ((mButton)findViewById(R.id.button2)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_TWO]);
194        ((mButton)findViewById(R.id.button3)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_THREE]);
        ((mButton)findViewById(R.id.button4)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_FOUR]);
196        ((mButton)findViewById(R.id.button5)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_FIVE]);
        ((mButton)findViewById(R.id.button6)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_SIX]);
198        ((mButton)findViewById(R.id.button7)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_SEVEN]);
        ((mButton)findViewById(R.id.button8)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_EIGHT]);
200        ((mButton)findViewById(R.id.button9)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_NINE]);
        ((mButton)findViewById(R.id.button_chanp)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_CHANP]);
202        ((mButton)findViewById(R.id.button_chanm)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_CHANM]);
        ((mButton)findViewById(R.id.button_power)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_POWER]);
204        ((mButton)findViewById(R.id.button_volp)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_VOLP]);
        ((mButton)findViewById(R.id.button_volm)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_VOLM]);
206        ((mButton)findViewById(R.id.button_source)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_SOURCE]);
        ((mButton)findViewById(R.id.button_mute)).mSetEnabled(
            [+] buttonsEnabled[BUTTON_MUTE]);

208    }

```

```

210 void setButtonsProgrammed(boolean programmed,
    [+]SharedPreferences.Editor editor){
212     editor.putBoolean("button_0_programmed", programmed);
    editor.putBoolean("button_1_programmed", programmed);
214     editor.putBoolean("button_2_programmed", programmed);
    editor.putBoolean("button_3_programmed", programmed);
216     editor.putBoolean("button_4_programmed", programmed);
    editor.putBoolean("button_5_programmed", programmed);
218     editor.putBoolean("button_6_programmed", programmed);
    editor.putBoolean("button_7_programmed", programmed);
220     editor.putBoolean("button_8_programmed", programmed);
    editor.putBoolean("button_9_programmed", programmed);
222     editor.putBoolean("button_volm_programmed", programmed);
    editor.putBoolean("button_volp_programmed", programmed);
224     editor.putBoolean("button_chanm_programmed", programmed);
    editor.putBoolean("button_chanp_programmed", programmed);
226     editor.putBoolean("button_source_programmed", programmed);
    editor.putBoolean("button_power_programmed", programmed);
228     editor.putBoolean("button_mute_programmed", programmed);
    editor.commit();
}

230 void setButtonsEnabled(boolean enabled){
232     ((mButton) findViewById(R.id.button0)).setEnabled(enabled
        [+]);
    ((mButton) findViewById(R.id.button1)).setEnabled(enabled
        [+]);
234     ((mButton) findViewById(R.id.button2)).setEnabled(enabled
        [+]);
    ((mButton) findViewById(R.id.button3)).setEnabled(enabled
        [+]);
236     ((mButton) findViewById(R.id.button4)).setEnabled(enabled
        [+]);
    ((mButton) findViewById(R.id.button5)).setEnabled(enabled
        [+]);
238     ((mButton) findViewById(R.id.button6)).setEnabled(enabled
        [+]);
    ((mButton) findViewById(R.id.button7)).setEnabled(enabled
        [+]);
240     ((mButton) findViewById(R.id.button8)).setEnabled(enabled
        [+]);
    ((mButton) findViewById(R.id.button9)).setEnabled(enabled
        [+]);
242     ((mButton) findViewById(R.id.button_chanp)).setEnabled(

```

```

        [+]enabled);
        ((mButton) findViewById(R.id.button_chanm)).setEnabled(
            [+]enabled);
244        ((mButton) findViewById(R.id.button_power)).setEnabled(
            [+]enabled);
        ((mButton) findViewById(R.id.button_volp)).setEnabled(
            [+]enabled);
246        ((mButton) findViewById(R.id.button_volm)).setEnabled(
            [+]enabled);
        ((mButton) findViewById(R.id.button_source)).setEnabled(
            [+]enabled);
248        ((mButton) findViewById(R.id.button_mute)).setEnabled(
            [+]enabled);
    }

    @Override
252    public void onNavigationDrawerItemSelected(int position) {
        // update the main content by replacing fragments
254        FragmentManager fragmentManager = getFragmentManager();
        fragmentManager.beginTransaction()
256            .replace(R.id.container, PlaceholderFragment.
                .newInstance(position + 1))
            .commit();
258    }

    public void onSectionAttached(int number) {
        switch (number) {
262            case 1:
                mTitle = getString(R.string.title_section1);
264                break;
            case 2:
                mTitle = getString(R.string.title_section2);
266                break;
            case 3:
                mTitle = getString(R.string.title_section3);
268                break;
270        }
272    }

    public void restoreActionBar() {
274        ActionBar actionBar = getActionBar();
276        actionBar.setNavigationMode(ActionBar.
            .NAVIGATION_MODE_STANDARD);
        actionBar.setDisplayShowTitleEnabled(true);
278        actionBar.setTitle(mTitle);
    }

```

```

    }
280
    @Override
282    public boolean onCreateOptionsMenu(Menu menu) {
        if (!mNavigationDrawerFragment.isDrawerOpen()) {
284            // Only show items in the action bar relevant to this
                // [+]screen
                // if the drawer is not showing. Otherwise, let the
                // [+]drawer
286            // decide what to show in the action bar.
            getMenuInflater().inflate(R.menu.main, menu);
288            restoreActionBar();
            return true;
290        }
        return super.onCreateOptionsMenu(menu);
292    }

    @Override
294    public void onClick(View v) {
296        switch(v.getId()){
            case R.id.button0:
298                if(recordSwitchBool){
                    sendRecord(BUTTON_ZERO);
300                    prefsEditor.putBoolean("button_0_programmed",
                        [+]true);
                } else {
302                    sendButton(BUTTON_ZERO);
                }
304                break;
            case R.id.button1:
306                if(recordSwitchBool){
                    sendRecord(BUTTON_ONE);
308                    prefsEditor.putBoolean("button_1_programmed",
                        [+]true);
                } else {
310                    sendButton(BUTTON_ONE);
                }
312                break;
            case R.id.button2:
314                if(recordSwitchBool){
                    sendRecord(BUTTON_TWO);
316                    prefsEditor.putBoolean("button_2_programmed",
                        [+]true);
                } else {
318                    sendButton(BUTTON_TWO);

```



```

    }
320     break;
    case R.id.button3:
322         if(recordSwitchBool){
            sendRecord(BUTTON_THREE);
324             prefsEditor.putBoolean("button_3_programmed",
                [+]true);
        } else {
326             sendButton(BUTTON_THREE);
        }
328     break;
    case R.id.button4:
330         if(recordSwitchBool){
            sendRecord(BUTTON_FOUR);
332             prefsEditor.putBoolean("button_4_programmed",
                [+]true);
        } else {
334             sendButton(BUTTON_FOUR);
        }
336     break;
    case R.id.button5:
338         if(recordSwitchBool){
            sendRecord(BUTTON_FIVE);
340             prefsEditor.putBoolean("button_5_programmed",
                [+]true);
        } else {
342             sendButton(BUTTON_FIVE);
        }
344     break;
    case R.id.button6:
346         if(recordSwitchBool){
            sendRecord(BUTTON_SIX);
348             prefsEditor.putBoolean("button_6_programmed",
                [+]true);
        } else {
350             sendButton(BUTTON_SIX);
        }
352     break;
    case R.id.button7:
354         if(recordSwitchBool){
            sendRecord(BUTTON_SEVEN);
356             prefsEditor.putBoolean("button_7_programmed",
                [+]true);
        } else {
358             sendButton(BUTTON_SEVEN);

```

```

    }
    break;
360 case R.id.button8:
    if(recordSwitchBool){
362         sendRecord(BUTTON_EIGHT);
364         prefsEditor.putBoolean("button_8_programmed",
            [+]true);
    } else {
366         sendButton(BUTTON_EIGHT);
    }
    break;
368 case R.id.button9:
    if(recordSwitchBool){
370         sendRecord(BUTTON_NINE);
372         prefsEditor.putBoolean("button_9_programmed",
            [+]true);
    } else {
374         sendButton(BUTTON_NINE);
    }
    break;
376 case R.id.button_chanm:
    if(recordSwitchBool){
378         sendRecord(BUTTON_CHANM);
380         prefsEditor.putBoolean("
            [+]button_chanm_programmed", true);
    } else {
382         sendButton(BUTTON_CHANM);
    }
    break;
384 case R.id.button_chanp:
    if(recordSwitchBool){
386         sendRecord(BUTTON_CHANP);
388         prefsEditor.putBoolean("
            [+]button_chanp_programmed", true);
    } else {
390         sendButton(BUTTON_CHANP);
    }
    break;
392 case R.id.button_mute:
    if(recordSwitchBool){
394         sendRecord(BUTTON_MUTE);
396         prefsEditor.putBoolean("button_mute_programmed
            [+]", true);
    } else {
398         sendButton(BUTTON_MUTE);
    }

```

```

    }
    break;
400 case R.id.button_volm:
    if(recordSwitchBool){
402         sendRecord(BUTTON_VOLM);
404         prefsEditor.putBoolean("button_volm_programmed
            [+]", true);
    } else {
406         sendButton(BUTTON_VOLM);
    }
    break;
408 case R.id.button_volp:
    if(recordSwitchBool){
410         sendRecord(BUTTON_VOLP);
412         prefsEditor.putBoolean("button_volp_programmed
            [+]", true);
    } else {
414         sendButton(BUTTON_VOLP);
    }
    break;
416 case R.id.button_power:
    if(recordSwitchBool){
418         sendRecord(BUTTON_POWER);
420         prefsEditor.putBoolean("
            [+]button_power_programmed", true);
    } else {
422         sendButton(BUTTON_POWER);
    }
    break;
424 case R.id.button_source:
    if(recordSwitchBool){
426         sendRecord(BUTTON_SOURCE);
428         prefsEditor.putBoolean("
            [+]button_source_programmed", true);
    } else {
430         sendButton(BUTTON_SOURCE);
    }
    case R.id.button_favs:
432         popupMenu.show();
434         break;
    }
436     prefsEditor.commit();
}
438
private void sendRecord(int button){

```

```

440         gpio.sendMacro("sendRecord/" + Integer.toString(button));
442     }
444     private void sendButton(int button){
446         gpio.sendMacro("sendButton/" + Integer.toString(button));
448     }
450     private void sendReset(){
452         gpio.sendMacro("sendReset");
454     }
456     @Override
458     protected void onResume(){
460         super.onResume();
462     }
464     @Override
466     protected void onPause(){
468         super.onPause();
470     }
472     @Override
474     public boolean onOptionsItemSelected(MenuItem item) {
476         //TODO change to selected channel
478         return false;
480     }
482     @Override
484     public boolean onOptionsItemSelected(MenuItem item) {
486         // Handle action bar item clicks here. The action bar will
488         // automatically handle clicks on the Home/Up button, so
490         // as you specify a parent activity in AndroidManifest.xml
492         // [+].
494         int id = item.getItemId();
496         if (id == R.id.action_settings) {
498             Intent i = new Intent(this, UserSettingsActivity.class
499             [+]);
500             startActivity(i);
502             return true;
504         }
506         return super.onOptionsItemSelected(item);
508     }
510     /**

```

```

482     * A placeholder fragment containing a simple view.
    */
484     public static class PlaceholderFragment extends Fragment {
        /**
486         * The fragment argument representing the section number
            [+]for this
        * fragment.
488         */
        private static final String ARG_SECTION_NUMBER = "
            [+]section_number";

490
        /**
492         * Returns a new instance of this fragment for the given
            [+]section
        * number.
494         */
        public static PlaceholderFragment newInstance(int
            [+]sectionNumber) {
496             PlaceholderFragment fragment = new PlaceholderFragment
                [+]();
            Bundle args = new Bundle();
498             args.putInt(ARG_SECTION_NUMBER, sectionNumber);
            fragment.setArguments(args);
500             return fragment;
        }

502
        public PlaceholderFragment() {
504
        }

506
        @Override
        public View onCreateView(LayoutInflater inflater,
            [+]ViewGroup container,
508             Bundle savedInstanceState) {
            View rootView = inflater.inflate(R.layout.
                [+]fragment_main, container, false);
510             return rootView;
        }

512
        @Override
514         public void onAttach(Activity activity) {
            super.onAttach(activity);
516             ((MainActivity) activity).onSectionAttached(
                getArguments().getInt(ARG_SECTION_NUMBER));
518         }
    }
}

```

520 }

Listing 3: `UserSettingsActivity.java`. This Activity displays settings when the “Settings” button is pressed in the main application window. Some preferences are hidden from the user.

```
package us.steveboyer.sdremote;

import android.os.Bundle;
import android.preference.PreferenceActivity;
import android.preference.PreferenceFragment;
import android.preference.PreferenceScreen;

/**
 * Created by steve on 4/1/15.
 */
public class UserSettingsActivity extends PreferenceActivity {

    public final static String MAIN_PREFERENCES = "MainPreferences
[+]";

    @Override
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        getFragmentManager().beginTransaction().replace(android.R.
[+]id.content, new MyPreferenceFragment()).commit();
    }

    public static class MyPreferenceFragment extends
[+]PreferenceFragment {
        @Override
        public void onCreate(final Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.preferences);
            PreferenceScreen screen = getPreferenceScreen();
            screen.removePreference(findPreference("
[+]button_0_programmed"));
            screen.removePreference(findPreference("
[+]button_1_programmed"));
            screen.removePreference(findPreference("
[+]button_2_programmed"));
            screen.removePreference(findPreference("
[+]button_3_programmed"));
            screen.removePreference(findPreference("
[+]button_4_programmed"));
            screen.removePreference(findPreference("
[+]button_5_programmed"));
```

```

32         screen.removePreference(findPreference("
           [+]button_6_programmed"));
           screen.removePreference(findPreference("
           [+]button_7_programmed"));
34         screen.removePreference(findPreference("
           [+]button_8_programmed"));
           screen.removePreference(findPreference("
           [+]button_9_programmed"));
36         screen.removePreference(findPreference("
           [+]button_volm_programmed"));
           screen.removePreference(findPreference("
           [+]button_volp_programmed"));
38         screen.removePreference(findPreference("
           [+]button_chanm_programmed"));
           screen.removePreference(findPreference("
           [+]button_chanp_programmed"));
40         screen.removePreference(findPreference("
           [+]button_mute_programmed"));
           screen.removePreference(findPreference("
           [+]button_source_programmed"));
42         screen.removePreference(findPreference("
           [+]button_power_programmed"));
           }
44     }
}

```

Listing 4: mGPIO.java. This class takes care of sending commands from the Android application.

```

1 package us.steveboyer.sdremote;

3 import android.util.Log;
  import com.androidhive.jsonparsing.JSONParser;

5 import org.json.JSONException;
7 import org.json.JSONObject;

9 /**
   * Created by steve on 4/15/15.
11 */
  public class mGPIO {
13     public static class mConnection {
           public String host;
15         public int port;
           public String username;
17         public String password;

```

```

19     public mConnection(String host) {
20         this.host = host;
21         this.port = 8000;
22         this.username = "webiopi";
23         this.password = "raspberry";
24     }
25
26     public mConnection(String host, int port, String user,
27         [+]String pass) {
28         this.host = host;
29         this.port = port;
30         this.username = user;
31         this.password = pass;
32     }
33 }
34
35 private mConnection conn;
36 private boolean connected;
37
38 public mGPIO(mConnection conn){
39     this.conn = conn;
40 }
41
42 /**
43  * outputSequence sends a sequence of bits to the specified
44  * GPIO port with the delay specified. The sequence is
45  * [+]specified
46  * as a string of 0 and 1 characters. It doesn't validate the
47  * string sent. This method works in an asynchronous way
48  * starting a new Thread to make the request and ends.
49  * <p>
50  * If there is any connection exceptions it generates
51  * a "ConnectionEvent", this is useful to detect if
52  * GPIO is disconnected from the target host.
53  * @param macro Name of macro
54  */
55 public void sendMacro(final String macro) {
56     final mGPIO self = this;
57     Thread t = new Thread(new Runnable() {
58         @Override
59         public void run() {
60             try {
61                 JSONParser parser = new JSONParser(conn.
62                     [+]username,

```



```

61         conn.password, conn.host);
        String url = "http://"
63             + conn.host + ":" + conn.port +
                "/macros/"+macro;
        Log.d("Sent", url);
65        JSONObject obj = parser.postJSONFromURL(url,
            [+]conn.port);

67        if (obj != null) {
            }
69        } catch (JSONException e) {

71        } catch (Exception e) {

73        }
75    }
    });
    t.start();
77 }
}

```

Listing 5: activity_main.xml. This file defines user interface elements of the main screen of the Android application.

```

<!-- A DrawerLayout is intended to be used as the top-level
[+]content view using match_parent for both width and height to
[+]consume the full space available. -->
2 <android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
4    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
6    android:layout_width="match_parent"
    android:layout_height="match_parent"
8    tools:context=".MainActivity">

10 <!-- As the main content view, the view below consumes the
    [+]entire
        space available using match_parent in both dimensions.
    [+]-->
12 <FrameLayout
    android:id="@+id/container"
14    android:layout_width="match_parent"
    android:layout_height="match_parent"
16    android:background="@layout/background" >

18 <RelativeLayout

```

```

20     android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_gravity="center_horizontal|top">
22
23     <us.steveboyer.sdremote.mButton
24         android:layout_width="@dimen/remote_button_width"
    android:layout_height="@dimen/remote_button_height
        [+] "
26         android:text="@string/text_button_1"
    android:id="@+id/button1"
28         android:layout_alignParentTop="true"
    android:layout_alignParentStart="true"
30         android:textSize="42dp"
    android:onClick="onClick"/>
32
33     <us.steveboyer.sdremote.mButton
34         android:layout_width="@dimen/remote_button_width"
    android:layout_height="@dimen/remote_button_height
        [+] "
36         android:text="2"
    android:id="@+id/button2"
38         android:layout_alignBottom="@+id/button1"
    android:layout_toEndOf="@+id/button1"
40         android:textSize="42dp"
    android:onClick="onClick"/>
42
43     <us.steveboyer.sdremote.mButton
44         android:layout_width="@dimen/remote_button_width"
    android:layout_height="@dimen/remote_button_height
        [+] "
46         android:text="3"
    android:id="@+id/button3"
48         android:layout_alignParentTop="true"
    android:layout_toEndOf="@+id/button2"
50         android:textSize="42dp"
    android:onClick="onClick"/>
52
53     <us.steveboyer.sdremote.mButton
54         android:layout_width="@dimen/remote_button_width"
    android:layout_height="@dimen/remote_button_height
        [+] "
56         android:text="4"
    android:id="@+id/button4"
58         android:layout_below="@+id/button1"
    android:layout_alignParentStart="true"

```

```

60         android:textSize="42dp"
        android:onClick="onClick"/>
62
63     <us.steveboyer.sdremote.mButton
64         android:layout_width="@dimen/remote_button_width"
        android:layout_height="@dimen/remote_button_height
        [+] "
65         android:text="5"
        android:id="@+id/button5"
66         android:layout_below="@+id/button1"
        android:layout_toEndOf="@+id/button1"
67         android:textSize="42dp"
        android:onClick="onClick"/>
68
69     <us.steveboyer.sdremote.mButton
70         android:layout_width="@dimen/remote_button_width"
71         android:layout_height="@dimen/remote_button_height
72         [+] "
73         android:text="6"
74         android:id="@+id/button6"
75         android:layout_below="@+id/button2"
76         android:layout_toEndOf="@+id/button2"
77         android:textSize="42dp"
78         android:onClick="onClick"/>
79
80     <us.steveboyer.sdremote.mButton
81         android:layout_width="@dimen/remote_button_width"
82         android:layout_height="@dimen/remote_button_height
83         [+] "
84         android:text="7"
85         android:id="@+id/button7"
86         android:layout_below="@+id/button5"
87         android:layout_alignParentStart="true"
88         android:textSize="42dp"
89         android:onClick="onClick"/>
90
91     <us.steveboyer.sdremote.mButton
92         android:layout_width="@dimen/remote_button_width"
93         android:layout_height="@dimen/remote_button_height
94         [+] "
95         android:text="8"
96         android:id="@+id/button8"
97         android:layout_toEndOf="@+id/button4"
98         android:layout_below="@+id/button4"
99         android:textSize="42dp"
100

```

```

102         android:onClick="onClick"/>

104     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
        android:layout_height="@dimen/remote_button_height
        [+] "
106         android:text="9"
        android:id="@+id/button9"
108         android:layout_toEndOf="@+id/button2"
        android:layout_below="@+id/button6"
110         android:layout_alignParentEnd="true"
        android:textSize="42dp"
112         android:onClick="onClick"/>

114     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
116         android:layout_height="@dimen/remote_button_height
        [+] "
        android:text="0"
118         android:id="@+id/button0"
        android:layout_toEndOf="@+id/button1"
120         android:layout_below="@+id/button8"
        android:textSize="42dp"
122         android:onClick="onClick"/>

124     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
126         android:layout_height="@dimen/remote_button_height
        [+] "
        android:text="Chan_+"
128         android:id="@+id/button_chanp"
        android:layout_alignParentStart="true"
130         android:layout_below="@+id/button8"
        android:textSize="30dp"
132         android:onClick="onClick"/>

134     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
136         android:layout_height="@dimen/remote_button_height
        [+] "
        android:text="Chan_-"
138         android:id="@+id/button_chanm"
        android:layout_alignParentStart="true"
140         android:layout_below="@+id/button_chanp"
        android:textSize="30dp"

```

```

142         android:onClick="onClick"/>

144     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
146         android:layout_height="@dimen/remote_button_height
            [+] "
        android:text="Favorites"
148         android:id="@+id/button_favs"
        android:layout_alignParentStart="true"
150         android:layout_below="@+id/button_chanm"
        android:textSize="23dp"
152         android:onClick="onClick"/>

154     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
156         android:layout_height="@dimen/remote_button_height
            [+] "
        android:text="Vol_+"
158         android:id="@+id/button_volp"
        android:layout_toEndOf="@+id/button2"
160         android:layout_below="@+id/button8"
        android:textSize="30dp"
162         android:onClick="onClick"/>

164     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
166         android:layout_height="@dimen/remote_button_height
            [+] "
        android:text="Vol_-"
168         android:id="@+id/button_volm"
        android:layout_alignParentRight="true"
170         android:layout_below="@+id/button_volp"
        android:textSize="30dp"
172         android:onClick="onClick"/>

174     <us.steveboyer.sdremote.mButton
        android:layout_width="@dimen/remote_button_width"
176         android:layout_height="@dimen/remote_button_height
            [+] "
        android:text="Mute"
178         android:id="@+id/button_mute"
        android:layout_alignParentRight="true"
180         android:layout_below="@+id/button_volm"
        android:textSize="30dp"
182         android:onClick="onClick"/>

```

```

184         <us.steveboyer.sdremote.mButton
186             android:layout_width="@dimen/remote_button_width"
             android:layout_height="@dimen/remote_button_height
               [+] "
188             android:text="Power"
             android:id="@+id/button_power"
190             android:layout_toRightOf="@+id/button7"
             android:layout_below="@+id/button0"
192             android:textSize="30dp"
             android:onClick="onClick"/>
194
196         <us.steveboyer.sdremote.mButton
             android:layout_width="@dimen/remote_button_width"
             android:layout_height="@dimen/remote_button_height
               [+] "
198             android:text="Source"
             android:id="@+id/button_source"
200             android:layout_toRightOf="@+id/button7"
             android:layout_below="@+id/button_power"
202             android:textSize="30dp"
             android:onClick="onClick"/>
204
206         <Switch
             android:layout_width="wrap_content"
             android:layout_height="@dimen/remote_button_height
               [+] "
208             android:text="Record_Mode"
             android:id="@+id/switch_record"
210             android:layout_alignParentBottom="true"
             android:layout_centerHorizontal="true"
212             android:textColor="#FFFFFF" />
214     </RelativeLayout>
    </FrameLayout>
216
218    <!-- android:layout_gravity="start" tells DrawerLayout to
        [+]treat
        this as a sliding drawer on the left side for left-to-
        [+]right
        languages and on the right side for right-to-left
        [+]languages.
        If you're not building against API 17 or higher, use
        android:layout_gravity="left" instead.-->

```

```

222 <!-- The drawer is given a fixed width in dp and extends the
    [+] full height of
    the container. -->
224 <fragment android:id="@+id/navigation_drawer"
    android:layout_width="@dimen/navigation_drawer_width"
226 android:layout_height="match_parent"
    android:layout_gravity="start"
228 android:name="us.steveboyer.sdremote.
    [+] NavigationDrawerFragment"
    tools:layout="@layout/fragment_navigation_drawer" />
230
</android.support.v4.widget.DrawerLayout>

```

Listing 6: preferences.xml. This file defines all of the Android application's preferences.

```

1 <?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/
    [+] res/android"
3     android:layout_width="wrap_content"
    android:layout_height="wrap_content">
5     <EditTextPreference
        android:key="@string/key_pref_ip"
7         android:title="@string/pref_ip"
        android:summary="@string/pref_ip_summ"
9         android:defaultValue="@string/pref_ip_default"
        android:layout_height="wrap_content"
11        android:layout_width="wrap_content" />
    <EditTextPreference
13        android:key="@string/key_pref_port"
        android:title="@string/pref_port"
15        android:summary="@string/pref_port_summ"
        android:defaultValue="@string/pref_port_default"
17        android:layout_height="wrap_content" />
    <EditTextPreference
19        android:key="@string/key_pref_pass"
        android:title="@string/pref_pass"
21        android:summary="@string/pref_pass_summ"
        android:defaultValue="@string/pref_pass_default"
23        android:layout_height="wrap_content" />
    <EditTextPreference
25        android:key="@string/key_pref_user"
        android:title="@string/pref_user"
27        android:summary="@string/pref_user_summ"
        android:defaultValue="@string/pref_user_default"
29        android:layout_height="wrap_content" />
    <CheckBoxPreference

```

```

31         android:key="@string/key_pref_reset_buttons"
        android:title="@string/pref_reset_buttons"
33         android:summary=""
        android:defaultValue="false"
35         android:layout_height="wrap_content" />
<ListPreference
37     android:key="pref_fav_button"
        android:title="@string/pref_fav_button"
39     android:summary="" />
<CheckBoxPreference
41     android:key="button_0_programmed"
        android:defaultValue="false" />
43 <CheckBoxPreference
        android:key="button_1_programmed"
45     android:defaultValue="false" />
<CheckBoxPreference
47     android:key="button_2_programmed"
        android:defaultValue="false" />
49 <CheckBoxPreference
        android:key="button_3_programmed"
51     android:defaultValue="false" />
<CheckBoxPreference
53     android:key="button_4_programmed"
        android:defaultValue="false" />
55 <CheckBoxPreference
        android:key="button_5_programmed"
57     android:defaultValue="false" />
<CheckBoxPreference
59     android:key="button_6_programmed"
        android:defaultValue="false" />
61 <CheckBoxPreference
        android:key="button_7_programmed"
63     android:defaultValue="false" />
<CheckBoxPreference
65     android:key="button_8_programmed"
        android:defaultValue="false" />
67 <CheckBoxPreference
        android:key="button_9_programmed"
69     android:defaultValue="false" />
<CheckBoxPreference
71     android:key="button_chanm_programmed"
        android:defaultValue="false" />
73 <CheckBoxPreference
        android:key="button_chanp_programmed"
75     android:defaultValue="false" />

```



```

77     <CheckBoxPreference
        android:key="button_power_programmed"
        android:defaultValue="false" />
79     <CheckBoxPreference
        android:key="button_volm_programmed"
81     android:defaultValue="false" />
    <CheckBoxPreference
83     android:key="button_volp_programmed"
        android:defaultValue="false" />
85     <CheckBoxPreference
        android:key="button_mute_programmed"
87     android:defaultValue="false" />
    <CheckBoxPreference
89     android:key="button_source_programmed"
        android:defaultValue="false" />
91 </PreferenceScreen>

```

Listing 7: strings.xml. This file defines Strings used in the Android application. They are defined here to make translating easier.

```

1 <?xml version="1.0" encoding="utf-8"?>
  <resources>
3     <string name="app_name">SD Remote</string>
    <string name="title_section1">Living Room</string>
5     <string name="title_section2">Section 2</string>
    <string name="title_section3">Section 3</string>
7     <string name="navigation_drawer_open">Open navigation drawer</
        [+>string>
    <string name="navigation_drawer_close">Close navigation drawer
        [+></string>
9     <string name="action_connect">Connect</string>
    <string name="action_settings">Settings</string>
11    <string name="text_button_1">1</string>
    <string name="text_button_2">String</string>
13    <string name="pref_ip">IP Address</string>
    <string name="pref_ip_summ">IP Address of Hub</string>
15    <string name="pref_ip_default">192.168.1.1</string>
    <string name="pref_pass">Password</string>
17    <string name="pref_pass_summ">Password for WebIOPi on Hub</
        [+>string>
    <string name="pref_pass_default">raspberrry</string>
19    <string name="pref_port">Port Number</string>
    <string name="pref_port_summ">Port Number for WebIOPi on Hub</
        [+>string>
21    <string name="pref_port_default">8000</string>
    <string name="pref_user">Username</string>

```

```

23     <string name="pref_user_summ">Username for WebIOPi on Hub</
        [+]string>
    <string name="pref_user_default">webiopi</string>
25     <string name="pref_reset_buttons">Reset Recorded Buttons</
        [+]string>
    <string name="pref_reset_child">Reset Child Remote Codes</
        [+]string>
27     <string name="pref_fav_button">Set Favorite Channels</string>
    <string name="key_pref_user">pref_user</string>
29     <string name="key_pref_pass">pref_pass</string>
    <string name="key_pref_ip">pref_ip</string>
31     <string name="key_pref_port">pref_port</string>
    <string name="key_pref_fav">pref_fav_button</string>
33     <string name="key_pref_reset_buttons">pref_reset_buttons</
        [+]string>
    <color name="blue">#2F4F4F</color>
35 </resources>

```

Listing 8: `mButton.java`. This class creates an Android button that can also easily be toggled to unclickable and transparent.

```

1 package us.steveboyer.sdremote;

3 import android.content.Context;
  import android.util.AttributeSet;
5 import android.widget.Button;

7 /**
   * Created by steve on 4/9/15.
9  */
  public class mButton extends Button {
11     private boolean enabled = false;

13     public mButton(Context context){
        super(context);
15     }

17     public mButton(Context context, AttributeSet attrs) {
        super(context, attrs);
19     }

21     public mButton(Context context, AttributeSet attrs, int
        [+]defStyle) {
        super(context, attrs, defStyle);
23     }

```

```

25     public boolean mIsEnabled(){
        return enabled;
27     }

29     public void mSetEnabled(Boolean enabled){
        this.enabled = enabled;
31         if(enabled){
            this.setClickable(true);
33             this.setAlpha((float) 1.0);
        } else {
35             this.setClickable(false);
            this.setAlpha((float) 0.25);
37         }
        }
39     }
}

```

Hub

Listing 9: gpioV2.py. This is a Python script for starting an HTTP server and listening for commands.

```

#!/usr/bin/env python
2 import webiopi
import time
4 import vw
import pigpio

6
GPIO = webiopi.GPIO

8
pi = pigpio.pi()
10 BPS = 1000
RX = 11
12 TX = 25
rx = vw.tx(pi, RX, BPS)
14 tx = vw.tx(pi, TX, BPS)
STATUS = 4
16 NUM_SENDS = 1

18 webiopi.setDebug()

20
def setup():
22     webiopi.debug("Setup...")

```

```

GPIO.setFunction(STATUS, GPIO.OUT)
24 GPIO.digitalWrite(STATUS, GPIO.HIGH)

26
def loop():
28     webiopi.sleep(5)

30
def destroy():
32     GPIO.digitalWrite(STATUS, GPIO.LOW)

34
def blink(numTiimes):
36     for x in range(0, numTiimes):
        GPIO.digitalWrite(STATUS, GPIO.LOW)
38         webiopi.sleep(0.2)
        GPIO.digitalWrite(STATUS, GPIO.HIGH)
40         webiopi.sleep(0.2)

42
@webiopi.macro
44 def sendButton(button):
    blink(1)
46     while not tx.ready():
        time.sleep(0.1)
48     time.sleep(0.2)
    webiopi.debug("Sending␣" + str(0) + button)
50     tx.put(str(0) + button)

52
@webiopi.macro
54 def sendRecord(button):
    blink(2)
56     while not tx.ready():
        time.sleep(0.1)
58     time.sleep(0.2)
    webiopi.debug("Sending␣" + str(1) + button)
60     tx.put(str(1) + button)

62
@webiopi.macro
64 def sendReset():
    blink(3)
66     while not tx.ready():
        time.sleep(0.1)

```

```

68     blink(2)
        time.sleep(0.2)
70     webiopi.debug("Sending␣" + str(2))
        tx.put(str(2))
72     time.sleep(1)

```

Child

Listing 10: `childuino.ino`. The program responsible for controlling the *child* device.

```

#include <VirtualWire.h>
2 #include <IRremote.h>
#include <EEPROMvar.h>
4 #include <EEPROMex.h>

6 #define CHILD_ID 0x30;
#define SBUTTON 0x30
8 #define SRECORD 0x31
#define SRESET 0x32
10 #define NULL 0x21

12 //const int maxAllowedWrites = 20;
const int memBase = 120;
14
const int IR_IN = 10;
16 const int STATUS_PIN = 11;
const int VW_TRANSMIT_PIN = 12;
18 const int VW_RECEIVE_PIN = 13;
const int VW_TRANSMIT_EN_PIN = 4; // No wire here
20 const int WAIT_MAX = 5; // Max time to wait for IR input

22 IRrecv irrecv(IR_IN);
IRsend irsend;
24
decode_results results;
26
int codeType = -1;
28 unsigned long codeValue;
unsigned int rawCodes[RAWBUF];
30 int codeLen;
int toggle = 0;
32 const int NUM_IR_CODES = 20;
unsigned long IR_codes[NUM_IR_CODES];
34 int codesProgrammed[NUM_IR_CODES];

```

```

    int addrCodesProgrammed = 450;
36 int address = 0;
    int lastButtonState;
38 const int drSize = 20;
    const int numCodes = 20;
40 int addresses[numCodes];

42 void writeVar(EEPROMVar<decode_results> &eepromvar) {
    eepromvar.save();
44 }

46 void generateAddresses(){
    for(int i = 0; i < numCodes; i++){
48         addresses[i] = drSize*i;
    }
50 }

52 void setup()
{
54     // Delay
    delay(1000);

56     // Init status
    pinMode(STATUS_PIN, OUTPUT);
    blink(STATUS_PIN, 1);
60     // Init serial
    Serial.begin(9600);

62     // Init VirtualWire
64     vw_set_tx_pin(VW_TRANSMIT_PIN);
    vw_set_rx_pin(VW_RECEIVE_PIN);
66     vw_set_ptt_pin(VW_TRANSMIT_EN_PIN);
    //vw_set_ptt_inverted(true); // Required for DR3100
68     vw_setup(1000);           // Bits per sec
    vw_rx_start();             // Start the receiver PLL running
70     // Init IR in
    irrecv.enableIRIn();
72     EEPROM.setMemPool(memBase, EEPROMSizeUno);
    //EEPROM.setMaxAllowedWrites(maxAllowedWrites);
74     generateAddresses();
    for(int i = 0; i<numCodes; i++){
76         codesProgrammed[i] = 0;
        Serial.println("setup" + i);
78     }
    if(EEPROM.isReady()){

```

```

80     EEPROM.readBlock(addrCodesProgrammed, codesProgrammed);
    }
82 }

84 void blink(int pin, int ntimes){
    for(int i = 0; i < ntimes; i++){
86         digitalWrite(STATUS_PIN, LOW);
        delay(100);
88         digitalWrite(STATUS_PIN, HIGH);
        delay(100);
90     }
    }
92

void loop()
94 {
    uint8_t buf[VW_MAX_MESSAGE_LEN];
96     uint8_t buflen = VW_MAX_MESSAGE_LEN;
    if (vw_get_message(buf, &buflen)) // Non-blocking
98     {
        int i;
100        digitalWrite(STATUS_PIN, HIGH); // Flash a light to show
            [+]received good message
            // Message with a good checksum received, dump it.
102        Serial.print("Got:␣");
        for (i = 0; i < buflen; i++) {
104            Serial.print(buf[i], HEX);
            Serial.print('␣');
106        }
        digitalWrite(STATUS_PIN, LOW);
108        if (buf[0] == SRECORD) {
            blink(STATUS_PIN, 1);
110            Serial.println("Record␣mode");
            int button = 0;
112            int b1 = buf[2];
            b1 = b1 - 48;
114            int b2 = buf[3];
            b2 = b2 - 48;
116            if(b1 == 1){
                button = b1*10 + b2;
118            } else {
                button = b2;
120            }
            Serial.println("Button:␣" + button);
122            int currentAddress = addresses[button];
            irrecv.enableIRIn();

```

```

124     boolean wait = false;
        int time = 0;
126     digitalWrite(STATUS_PIN, HIGH);
        while(!irrecv.decode(&results)){
128         if(!wait){
            wait = true;
130             Serial.println("Waiting...");
        }
132     }
        digitalWrite(STATUS_PIN, LOW);
        storeCode(&results);
134     EEPROMVar<decode_results> eepResults = results;
        eepResults.setAddress(memBase + currentAddress);
136     writeVar(eepResults);
        irrecv.resume();
138 } else if (buf[0] == SBUTTON) {
140     blink(STATUS_PIN, 2);
        int button = 0;
142     Serial.println("Button mode");
        int b1 = buf[2];
144     b1 = b1 - 48;
        int b2 = buf[3];
146     b2 = b2 - 48;
        if(b1 == 1){
148         Serial.println("b1");
            button = b1*10 + b2;
150     } else {
        Serial.println("!b1");
152         button = b1;
    }
154     Serial.print("Button: ");
        Serial.println(button);
156     int currentAddress = addresses[button];
        EEPROM.readBlock<decode_results>(memBase +
            [+]currentAddress, results);
158     int iaddress = memBase + currentAddress;
        Serial.println(iaddress);
160     sendCode(false);
        delay(100);
162 } else if(buf[0] == SRESET) {
        Serial.println("Reset");
164     blink(STATUS_PIN, 3);
        for(int i = 0; i<numCodes; i++){
166         codesProgrammed[i] = 0;
        }

```



```

168     } else {
169         Serial.println("Bad_receive:_" + buf[0]);
170         blink(STATUS_PIN, 4);
171     }
172 }
173 }
174
175 // Stores the code for later playback
176 // Most of this code is just logging
177 void storeCode(void* v){
178     decode_results *results = (decode_results *)v;
179 //void storeCode(decode_results *results) {
180
181     //codeType = results->decode_type;
182     int count = results->rawlen;
183     if (codeType == UNKNOWN) {
184         Serial.println("Received_unknown_code,_saving_as_raw");
185         codeLen = results->rawlen - 1;
186         // To store raw codes:
187         // Drop first value (gap)
188         // Convert from ticks to microseconds
189         // Tweak marks shorter, and spaces longer to cancel out IR
190         //      [+]receiver distortion
191         for (int i = 1; i <= codeLen; i++) {
192             if (i % 2) {
193                 // Mark
194                 rawCodes[i - 1] = results->rawbuf[i]*USECPERTICK -
195                     [+]MARK_EXCESS;
196                 Serial.print("_m");
197             }
198             else {
199                 // Space
200                 rawCodes[i - 1] = results->rawbuf[i]*USECPERTICK +
201                     [+]MARK_EXCESS;
202                 Serial.print("_s");
203             }
204             Serial.print(rawCodes[i - 1], DEC);
205         }
206         Serial.println("");
207     }
208     else {
209         if (codeType == NEC) {
210             Serial.print("Received_NEC:_");
211             if (results->value == REPEAT) {
212                 // Don't record a NEC repeat value as that's useless.

```

```

210     Serial.println("repeat; ignoring.");
        return;
212     }
    }
214     else if (codeType == SONY) {
        Serial.print("Received_SONY: ");
216     }
    else if (codeType == RC5) {
218         Serial.print("Received_RC5: ");
    }
220     else if (codeType == RC6) {
        Serial.print("Received_RC6: ");
222     }
    else {
224         Serial.print("Unexpected_codeType ");
        Serial.print(codeType, DEC);
226         Serial.println("");
    }
228     Serial.println(results->value, HEX);
    codeValue = results->value;
230     codeLen = results->bits;
}
232 }

234 void sendCode(int repeat) {
    if (codeType == NEC) {
236         if (repeat) {
            irsend.sendNEC(REPEAT, codeLen);
238             Serial.println("Sent_NEC_repeat");
        }
240         else {
            irsend.sendNEC(codeValue, codeLen);
242             Serial.print("Sent_NEC ");
            Serial.println(codeValue, HEX);
244         }
    }
246     else if (codeType == SONY) {
        irsend.sendSony(codeValue, codeLen);
248         Serial.print("Sent_Sony ");
        Serial.println(codeValue, HEX);
250     }
    else if (codeType == RC5 || codeType == RC6) {
252         if (!repeat) {
            // Flip the toggle bit for a new button press
254             toggle = 1 - toggle;

```

```

    }
256 // Put the toggle bit into the code to send
    codeValue = codeValue & ~(1 << (codeLen - 1));
258 codeValue = codeValue | (toggle << (codeLen - 1));
    if (codeType == RC5) {
260         Serial.print("Sent_RC5_");
        Serial.println(codeValue, HEX);
262         irsend.sendRC5(codeValue, codeLen);
    }
264 else {
        irsend.sendRC6(codeValue, codeLen);
266         Serial.print("Sent_RC6_");
        Serial.println(codeValue, HEX);
268     }
}
270 else if (codeType == UNKNOWN /* i.e. raw */) {
    // Assume 38 KHz
272     irsend.sendRaw(rawCodes, codeLen, 38);
    Serial.println("Sent_raw");
274 }
}

```