LiFi

Kianoosh.Salami Justin.Cotton Elush.Shirazpour

Advisor: Bryan.Mealy

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT
COMPUTER ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2012

ABSTRACT

This paper will cover the steps necessary to design, build, and manage networkable smart light switches over WiFi via a mobile application. It contains a microcontroller running a real time operating system, a WiFi module, a Java server with SQLite databasing, and a mobile phone application.

LyFi is designed to turn on/off and dim household lights while only requiring the same installation procedure as a less capable dimmer switch. Once installed and connected to a network, LyFi will self-discover and configure. The server automatically propogates a database of nodes where any mobile device running an application can receive an updated controlable list. Finally, LyFi's hardware is not limited to only dimming lights, it can be modified to control other applicances.

TABLE OF CONTENTS

LIST OF FIGURES

This page intentially left blank.

ACKNOWLEGMENTS

This senior project would not have been possible without the support of many people. We would like to express our gratitude to our advisor Bryan Mealy who was abundantly helpful and offered invaluable assistance, support, and guidance. We would also like to thank Downtown San Luis Obispo for allowing us to hold many late night meetings and providing tasteful refreshments.

INTRODUCTION

Our senior project aimed to create a device that can be controlled/managed via WiFi connection. We wanted to create hardware capable of connecting to a variety of electronic devices and allowing the user to control that device remotely with a phone application. We choose to use WiFi since it can be found in almost all homes or businesses in the United States as well as the rest of the world.

Before entering this project, we knew that we needed to have a WiFi module, microcontroller, server, and a mobile application to have a successful product. We created a PCB capable of dealing with voltages up to 120V and supply DC voltages to the microcontroller and WiFi module. We chose the ATMega 328P microcontroller which runs a real time operating system that allows the processor to easily switch between different tasks. Tasks in our RTOS included the parser, switch, and encoder.

The server also runs SQLite to log all node and mobile client connections created. We also needed to open a connection port between the server and WiFi module. To ensure successful message passing, a UDP packet broadcasts to find the server. Once found, a TCP socket connection between the server and WiFi module happens. Once that completes, a parser initializes to allow the WiFi module and microcontroller to interpret commands with one another. The microcontroller sends commands to the WiFi module to perform various actions.

The next step was to create an application to control the selected device via WiFi. The application communicates with the server and can send values to the WiFi module via the server. All these parts of the project were considered with the “end user” in mind. We wanted the project to be easy to set up, use, and troubleshoot by anyone. We also took size in consideration and created hardware as small as possible to allow the boards to fit in a standard (1-gang) electrical box. The following describes the intricate details involving each aspect of the senior project. See figure 1 for the overall block diagram.

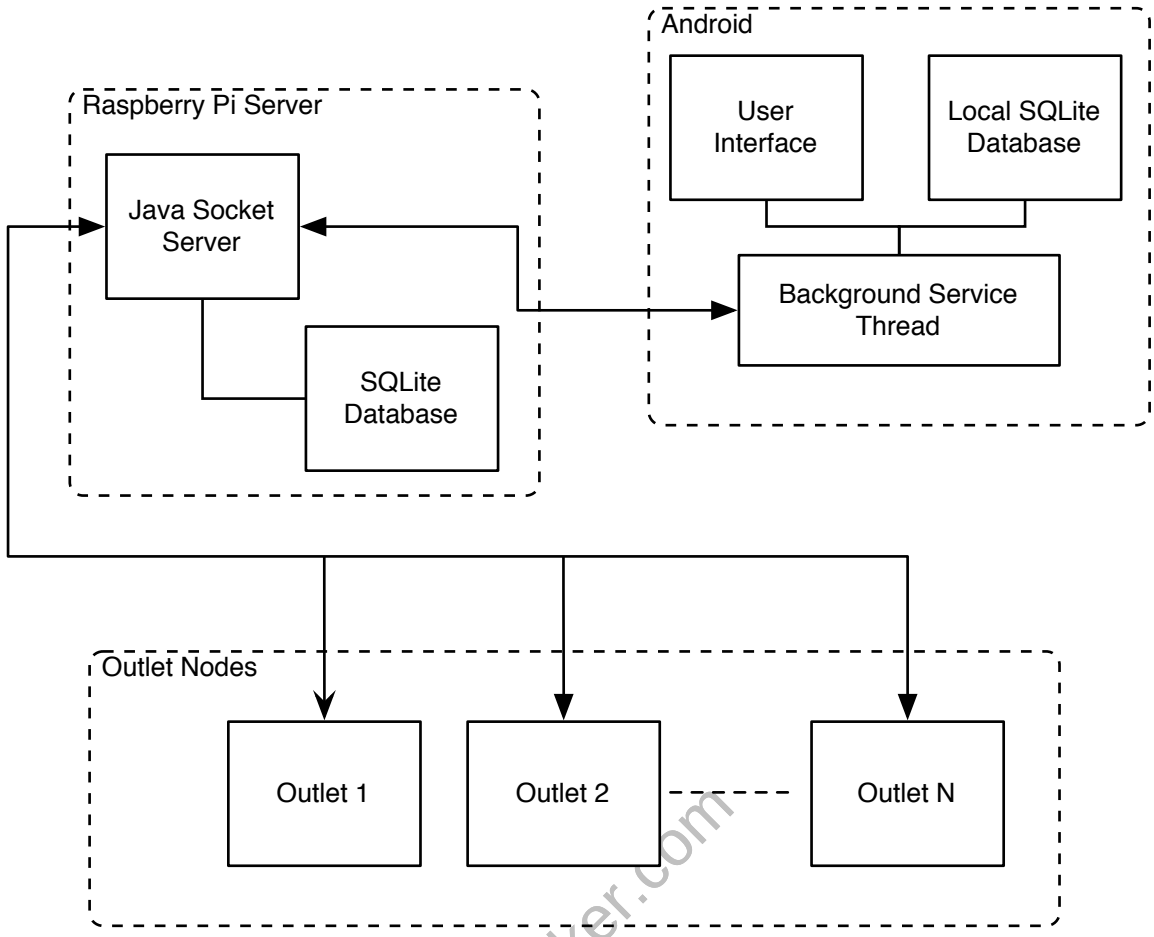


Figure 1 Overall System Block Diagram

FREERTOS INTEGRATION

FreeRTOS is a real-time operating system for embedded devices. We choose to use a real time operating system because it allows multiple processes to run *simultaneously*, to perform simple task switching, and to have the ability to save data in queues. FreeRTOS' scheduler relies on 'ticks' for keeping track of time; a hardware timer controls these ticks. We utilize the 16-bit timer. The FreeRTOS source files allow relatively easy porting. The only file needing adjustment is `port.c`. The modified macros and registers are below, figure 2.

```
#define portCLEAR_COUNTER_ON_MATCH              ( ( unsigned char ) _BV(WGM12) )
#define portPRESCALE_64                         ( ( unsigned char ) (_BV(CS11) | _BV(CS10)) )
#define portCLOCK_PRESCALER                     ( ( unsigned long ) 64 )
#define portCOMPARE_MATCH_A_INTERRUPT_ENABLE    ( ( unsigned char ) _BV(OCIE1A) )

/* Setup clock source and compare match behaviour. */
TCCR1A &= ~(_BV(WGM11) | _BV(WGM10));
ucLowByte = portCLEAR_COUNTER_ON_MATCH | portPRESCALE_64;
TCCR1B = ucLowByte;
/* Enable the interrupt - this is okay as interrupt are currently globally
disabled. */
ucLowByte = TIMSK1;
ucLowByte |= portCOMPARE_MATCH_A_INTERRUPT_ENABLE;
TIMSK1 = ucLowByte;
```

Figure 2 FreeRTOS Hardware Configuration

Finally, in `FreeRTOSConfig.h` the heap size must be adjusted to the corresponding SRAM size of the MCU. The Atmega328p has 2K (same as the Atmega323) of available SRAM therefore `configTOTAL_HEAP_SIZE` remains `1500`.

HARDWARE AND FIRMWARE

The goal of this project is to design hardware that can easily fit in a standard (1-gang) electrical box; consequently hardware size is a major priority. The microprocessor we choose is an Atmel Atmega328p as we had experience with Atmel chips utilizing FreeRTOS, its availability, and small footprint.

The next major hardware device is the WiFi chip. After much research, we decided on the GainSpan GS1011M WiFi module. The module is specifically designed to add WiFi to embedded systems where a complicated network stack is beyond the scope of the hardware. All communication between the WiFi module and MCU is over USART using specific AT (attention) commands. This allows executing complicated tasks (connecting to networks, enabling encryption, sending/receiving packets) via simple strings. The hardware used to switch and dim an AC light-bulb is explained in the dimmer section.

Finally, to power the embedded devices a small and efficient AC/DC 3.3V converter is needed. We picked a Recom RAC01-3.3SC switching regulator as it requires no external components, runs cool, and is sealed in a relatively small package.

DIMMER HARDWARE

Original dimmers diverted energy into an adjustable resistor to decrease the power given to a load. This method is not only inefficient, but dissipates a great deal of heat. Modern dimmers switch power on/off at high speed decreasing the amount of total available energy. To prevent flickering the light-bulb power needs to be switched off at the same time as the AC's wave zero-crossing, (figure 3) then turned on after a small delay (figture 4). The longer the delay, the less slice of AC energy is available to the light bulb, thus the dimmer the light appears. No delay after a zero crossing results in full brightness.

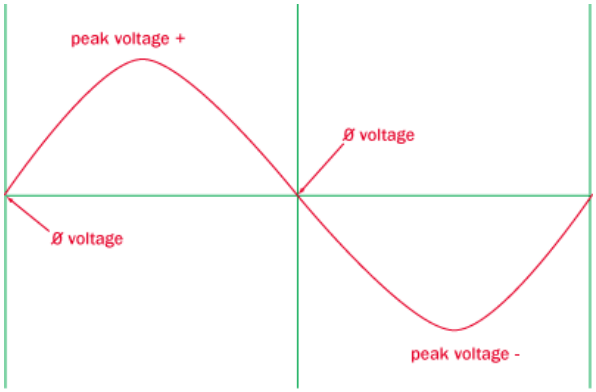


Figure 3 Zero-Crossing

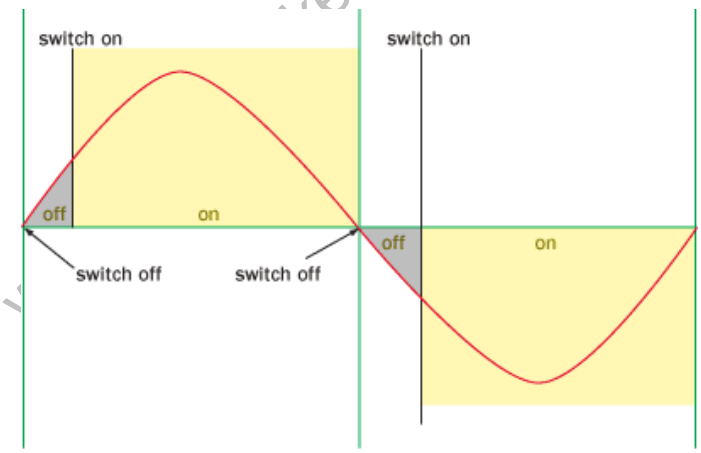


Figure 4 AC Slicing

There are two important circuits required: one for giving zero-crossing feedback to the microprocessor, and a second to safely switch AC power on and off (see APPENDIX A).

The zero-cross circuit works by feeding the AC source into resistors to drop the voltage. Next the AC signal is fed into a full bridge rectifier and converted into a DC signal. The

DC signal is fed into a 4N25 optocoupler where a phototransistor is switched on an off. The source of the transistor is fed into the microprocessor in parallel with a pull-up resistor with the drain of the transistor is connected to ground. When the AC voltage is not 0V the transistor is on, bringing the zero-cross signal to 0V. When the AC voltage passes through the zero crossing, the voltage is 0V into the optocoupler LED, turning off the transistor. There is no path to ground, and therefore no voltage drop across the pull-up resistor resulting in VCC on the zero-cross signal. This circuit provides a nice digital zero-cross detector and is connected to an interrupt-enabled pin, (see APPENDIX A).

To control the AC load digitally, we used a similar isolation circuit. The digital signal is fed into a MOC3021 optoisolator with triac out. Similar to the zero-cross circuit, when VCC is fed into the optoisolator the triac is turned on. The triac output controls a beefier BT136 triac capable of 4A.

FIRMWARE FOR DIMMER

Originally we used software delays to turn on and off the triac, however they were unreliable. We changed to use minimal software and timer0 in compare match mode our delays. When the circuit detects a zero-crossing interrupt (figure 5), the triac is turned off and the timer count is reset to zero. Once the timer count is equal to the compare register it triggers a separate ISR turning on the triac. These two ISRs simply slice of an AC signal allowing the light-bulb to dim.

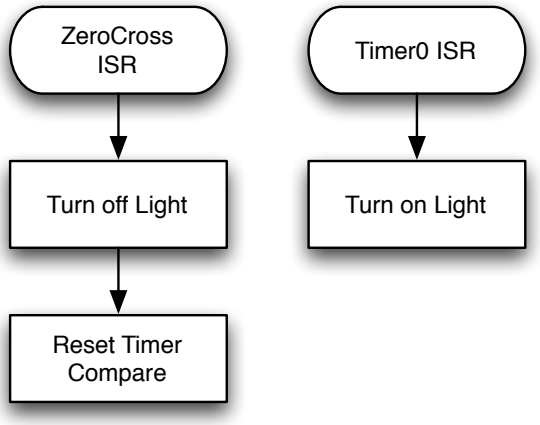


Figure 5 Dimmer Flow ISR Flow Chart

ENCODER

To control the light without the use of the mobile application, we used a rotary encoder, A rotary encoder is an electro-mechanical device that converts the angular position to digital code, called gray code. A potentiometer is limited in how many times it can be spun in one direction. An encoder was chosen instead of a potentiometer since we knew the application can also change the dimmer value. The encoder can change the value of the dimmer value relative to the app. The encoder can be pushed, like a switch, or rotate endlessly either clockwise or counterclockwise. For the senior project, pushing the encoder in will turn on/off the light and spinning it will cause the dimmer to increase/decrease light intensity. The encoder works by sensing are two signals, A and B, when spinning the encoder. Spinning in a certain direction causes a switching to occur (figure 6).

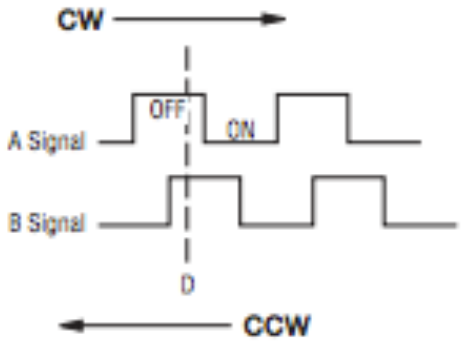


Figure 6 Encoder Pin Diagram

An interrupt service is called every time the encoder is spun. This interrupt activates a binary semaphore, in which the task is unblocked. As the encoder is being spun, its value is sent straight to the timer compare register, which is the dimmer value.

An interrupt was also created for every time the encoder switch is pushed. The interrupt would activate a binary semaphore, in which the task would be unblocked. Pushing it would either turn on/off the light.

An initial issue with the encoder controlling the dimmer value was that the values decrypted from the gray code would not cause a smooth transition for the dimmer value. Spinning the encoder too quickly would cause the encoder to freeze or skip values in between. To deal with this, a filtering capacitor was added (figure 7).

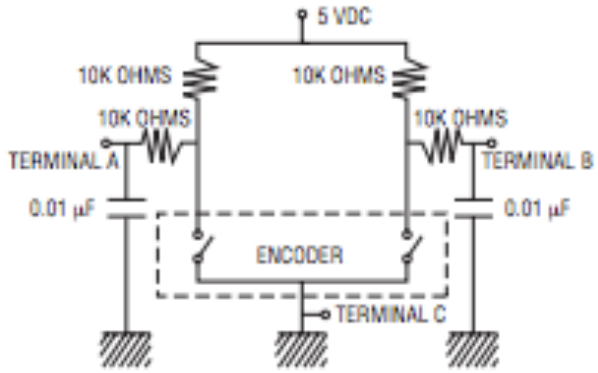


Figure 7 Encoder Filtering Circuit

The circuit was found on the encoder's datasheet and really helped. Spinning the encoder with the filtering capacitor made changing the dimmer value of the light significantly smoother. Figure 8 shows the encoder and switch block diagram.

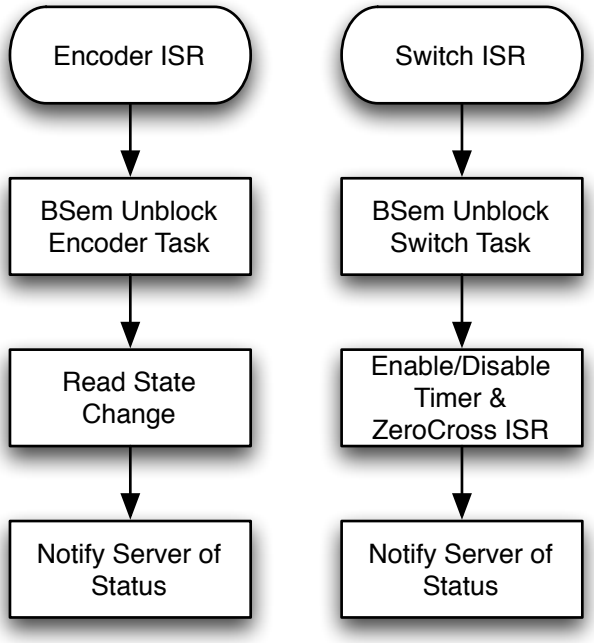


Figure 8 Encoder/Switch ISR Flow Chart

HARDWARE INTIALIZATION

When the hardware powers on, the firmware checks if it setup for relay mode, or dimmer mode. Once hardware is initialized, each of the tasks is initialized, ending with starting the FreeRTOS scheduler. Figure 9 shows the flow chart for the initialization.

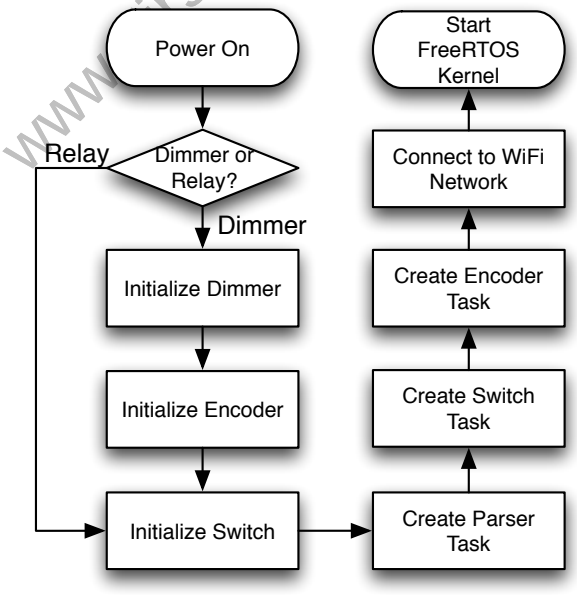


Figure 9 Hardware Initialization Flow Chart

PARSER

The parser first starts when the Atmega 328P receives data from the USART. Data from the USART is then placed into a serial queue and each byte from the queue is sent to the parser program. Data from the USART is never lost due to the microcontroller's ISR which pauses any progress done in the parser program to place new USART data into the serial queue. Figure 10 shows the parser flow chart.

`void makeBuf(unsigned char byte)`

Each individual byte from the serial queue is placed into a function called `makeBuf`. `makeBuf` is a multipurpose function in which it inserts certain bytes into a static array, called buf, and counts the length of the array. Due to RAM limitations, the array stores these bytes in the EEPROM. Before elements are stored in the array, `makeBuf` looks for the bytes Return, or ASCII 0x0D, and Newline, or ASCII 0x0A, in succession. The WiFi module always adds the Return and Newline before and after new data it receives. Once that occurs, `makeBuf` will record every byte in between into `buf`. The array continues to take in data until it records another `Return` and `Newline` in succession. Once this occurs, the array will only contain the bytes between the two pairs of Return and Newline. The program will continue into the function parser.

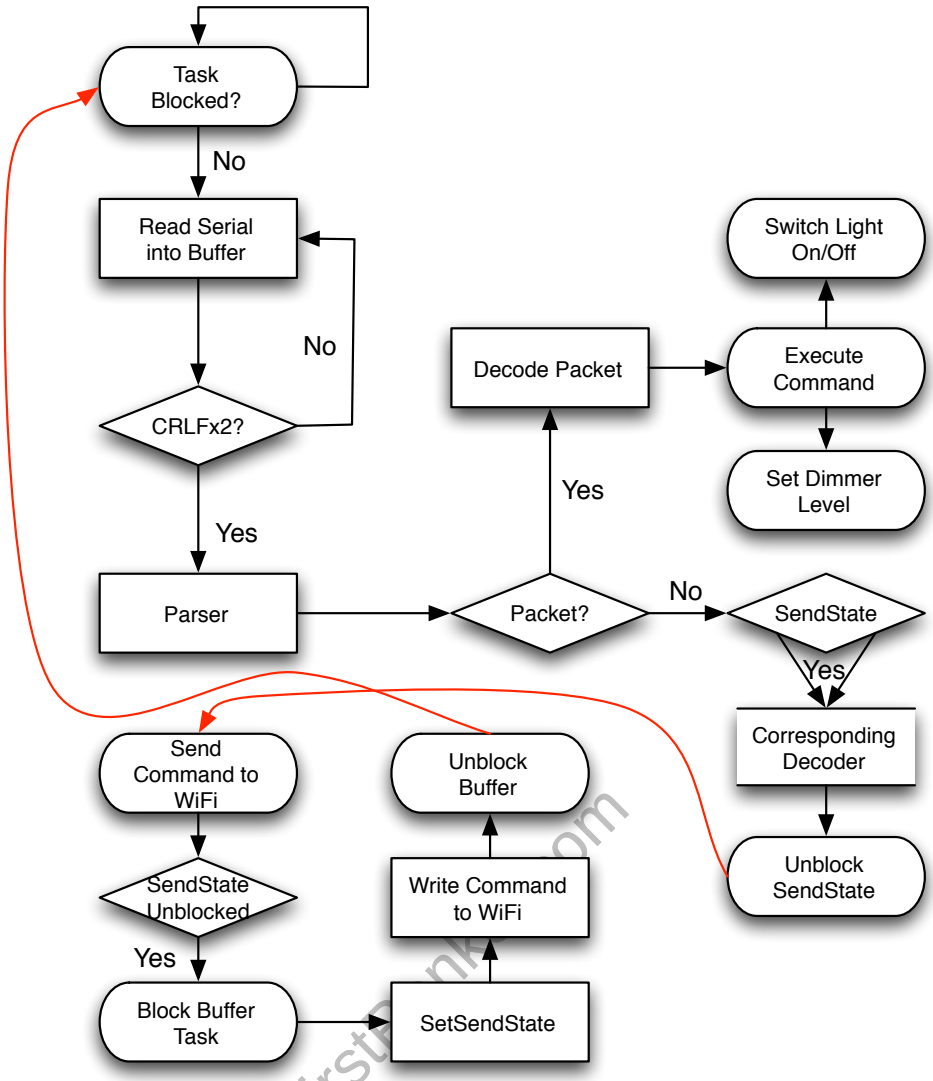


Figure 10 WiFi Command and Parser Flow Chart

`void parser()`

The parser function, in the parser task, determines what if the information in buf is a packet or not. If it is a packet, the buf's first two elements should be the ASCII values of Escape, 0x1B, and S and the buf's last two elements should be the ASCII values of Escape, 0x1B, and E. If this is the case, the program continues to a function called `serverPacket`. If this was not the case, the function continues onto a function called `sendStateCheck`.

void serverPacket()

serverPacket is function created to decode the message sent by the server. We created a certain format so we would be able to distinguish and interpret a message sent by the server. The format is decribed below:

byte 0: <ESC>
byte 1: <S>
byte 2: CID
byte 3: <$>
byte 4: ACTION
byte 5: LEN
byte 6-134: DATA
second to last byte: <ESC>
last byte: <E>

To explain in further detail, <ESC><S> and <ESC><E> are the first and last two bytes so that the parser can distinguish that this data is a server packet. CID represents the connection ID. It is the ASCII value of the connection ID created earlier. For example the ID 0 will be the ASCII value 0x30. The CID is obtained from the WiFi module to distinguish from different connections the WiFi module may have.
<$> is the special identifier so that we know the data received was certainly sent by our server.
ACTION represents the action the server would wish to take. The options are acknowledgment, connection, state, and type. Acknowledgment is represented with the character A. Connection is represented with the character C. State is represented with the character S. Type is represented with the character T. LEN represents the length of the upcoming data sent by the server. The ASCII value of the len is sent, much like CID. DATA represents the data that server wished to send to the microcontroller. Data can range from the dimmer value to be sent to the light or an on/off signal to be sent to the light.

`serverPacket` verifies that the data in the buf array has all the elements needed to be considered a correct message from the server. Since I know the correct format of a server packet, I know where each element should be located in the buf array. The function first records the second byte in buf, or the CID, in a local variable. The function then checks that the third byte is the same as the identifier. If the third byte of buf does not equal the server's identifier, the function returns back to the beginning of the parser program. That identifier is absolutely necessary to move forward with a server packet. If the byte is the identifier, it then checks the next byte to see what server action is being asked for. It will record the server action in a local variable. Data is all that is left in the packet. The length is captured from the fifth byte. The function moves onto another function called exec with the CID, ACTION, and LEN of the sever packet.

`void exec(int cid, unsigned char servAction, int len)`

`exec` is the function that executes the command sent from the server. Based on what the servAction is, it looks at the data sent by the server to perform the correct action for the correct CID. The function first starts of with a switch statement. The switch statements checks that the `servAction` is either acknowledgment, connection, state, or type. If it is neither of these, the data is ignored and the function will return to `makeBuf` with no changed made.

If the `servAction` is an acknowledgment, then we know that the data sent by the server is to switch light on/off or set the dimmer value. We check to make sure the `len` is 2, one byte for on/off and another byte for the dimmer value.

`void sendStateCheck ()`

`sendStateCheck` function is the function called by the parser function if the data in buf is not a packet. The `sendStateCheck` function contains all commands that are

needed to be called by the WiFi module to connect, send data, or close connection with the server. Based on the value of the sendState variable, a command is sent to the WiFi module. All WiFi commands are string literals stored in the EEPROM. EEPROM was used instead of program space due to RAM constraints. Based on the WiFi command, a certain response is expected to determine if the command was successful or not. A mutex semaphore unblocks the buffer allowing the decoder to interpret feedback. When a WiFi command returns, the buffer is then blocked, and the next command can be sent. The next `sendState` value is set and another command is sent to the WiFi module and the buffer is unblocked.

`encoder(int cid, char* str)`

The encoder function was used as a way to send commands to WiFi module to send to the server. By sending in the cid and the string implementing a certain command, a Return character was placed at the end. The reason for this was so that the WiFi module will execute the command. One specific command that was used was to let the server know the WiFi module was a node. The command for this was:


```
byte 0: <ESC>
byte 1: <S>
byte 2: CID
byte 3: <$>
byte 4: <T>
byte 5: <4>
byte 6-134: "NODE"
byte 7: <ESC>
byte 8: <E>
```

This command was sent from the WiFi module to the server to let the server know the WiFi module will be acting as a node, as opposed to a mobile client.

PCB LAYOUT

The PCB layout was done in Eagle CAD; APPENDIX A shows the schematic and layout. The ISP header is used to program the Atmega328 microprocessor. The GP27 jumper is used to update the firmware of the WiFi chip. Finally, the RX and TX connecting the microprocessor and WiFi chip have jumpers, these serve two purposes. First the WiFi chip firmware is updated through the RX and TX channels, but most importantly the jumpers allow the serial signals to be connected to a console for easing debugging. The various passive components utilized 0805 surface mount parts as they are small, but still can be soldered by hand.

SERVER

The original design of this project didn't contain a central server; however, after careful consideration and planning, we decided it would be beneficial to have a central hub for node and mobile connections. By implementing a server, a new mobile client could easily control each of the outlet nodes without having to re-initialize everything all over again. The server will hold a database of all connections, both outlet nodes and mobile clients, as well as maintain socket connections between itself (the server) and the nodes mobile clients.

We chose to use the *SQLite* engine to manage our database of all the node and mobile client connections. *SQLite* offers several key advantages: lightweight - less powerful hardware could be used for the server, multi-platform - this would allow us to choose our server hardware later as well as testing and debugging would be much easier because we could test on our own computers, and serverless - this makes creating and maintaining the database very easy to do.

The server code was written in *Java* because it is multiplatform (again, we didn't make a final decision on the server hardware), and it is simple to perform UDP and TCP socket connection. Another big reason was for easy compatibility between the server and the Android mobile client, which is also written in *Java.* This doesn't mean that using a *Java* server won't allow for communication between other mobile operating systems, it just made communication a lot easier.

As far as server hardware, we initially chose to use *Raspberry Pi*, which is an ARM processor running Linux and uses an external SD card for storage. It has network access via Ethernet which was a requirement for socket communication. However, this was a newly released product and a shipment date could not be guaranteed. Since our server would be able run on various platforms, we could develop, debug, and test it on a computer and then run it on the *Raspberry Pi* when it arrived.

SQLite Database

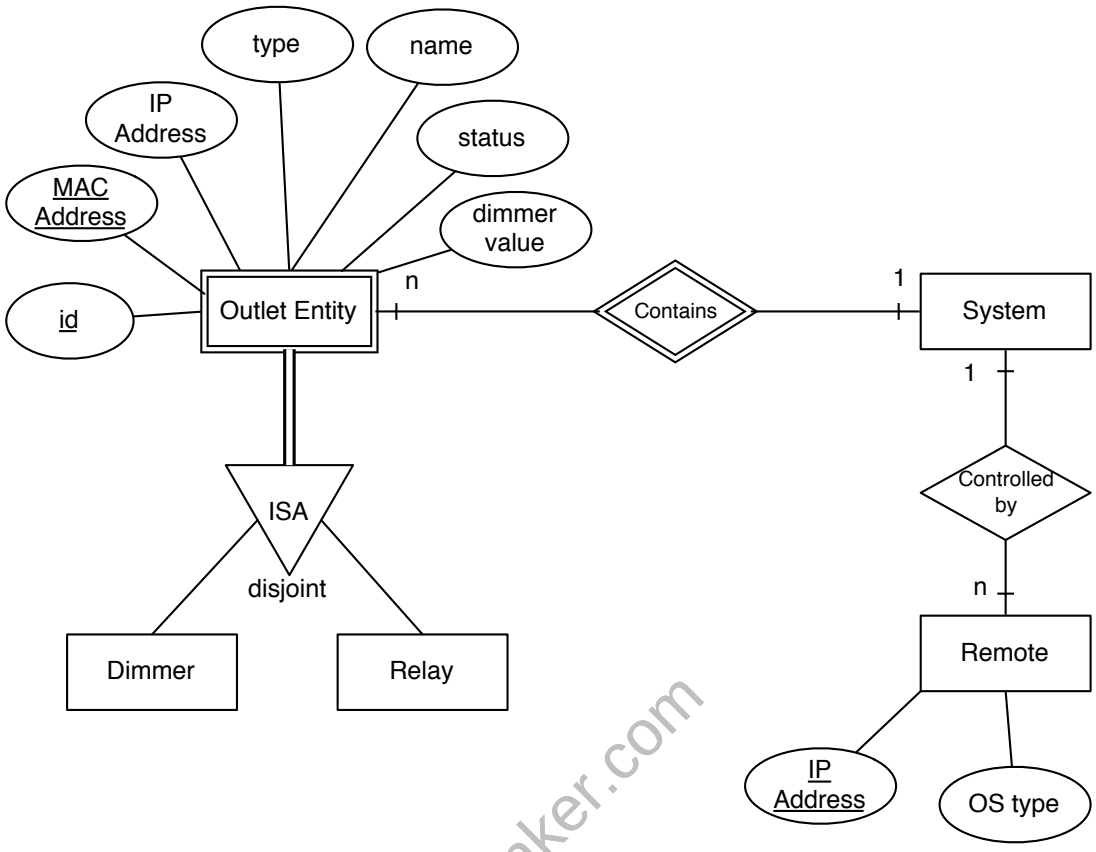


Figure 11 SQLite Entity Relationship Diagram

Figure 11 shows the basic schema for the database. Each *System* represents a house, which contains several *Outlets* and can be controlled by several *Remotes.* Each *Outlet* consists of an *ID,* which is used to simply identify a node in the database. A *MAC Address* is also used to identify a specific node as well as to guarantee that no node gets duplicated in the database. This helps for when the node receives a new IP address from a router's DHCP. The node's *IP Address* is used for debugging and testing to make sure the node is on the same network. The *Type* is used to determine if the node is a "dimmer" or a simple "relay." Which helps when the mobile application draws its view on the screen. The *Name* is just an arbitrary name for the outlet, like "Bryan's bedroom." This also allows the mobile application to display a meaningful name for each outlet. The *Status* simply indicates whether the outlet is ON ("1") or OFF

("0"). The *Dimmer Value* represents brightness of the light; obviously this is only used when the *Type* is a "dimmer." Both the *Status* and *Dimmer Value* attributes get changed when the user physically presses the switch on the outlet or changes its status from the mobile application. Lastly, each outlet node must only be considered as either a "Dimmer" or "Relay." Figure 12 shows is the code necessary to create a SQLite database with the attributes described before.

```
CREATE TABLE nodedetails (_id INTEGER PRIMARY KEY ASC,
                          _type TEXT NOT NULL,
                          _name TEXT,
                          _status INTEGER,
                           dimVal INTEGER,
```

Figure 12 SQLite Database

The mobile application is treated as the *Remote* in this configuration and each *Remote* has the ability to control all of the nodes in the *System.* The *Remote is simple;* it has an *IP Address* to identify each *Remote* and an *OS Type* to differentiate the software on the mobile device. The last attribute would be used for the addition of other mobile apps (ie. Apple's iOS)

In order for the server to communicate with the database and perform queries, a *SQLite* JDBC driver is required. This Java driver allows the server to make Java method calls which are executed just as if it is making native SQL queries or transactions. The server needs this driver in order to query the database for the entire node list as well as update out-of-sync nodes and mobile clients.

JAVA SERVER

In order for any server-node communication, each outlet node first has to connect to the local wireless network (see Conclusion). Next, the outlet node has to go through a connection initialization so it can communicate with the server, see figure 13.

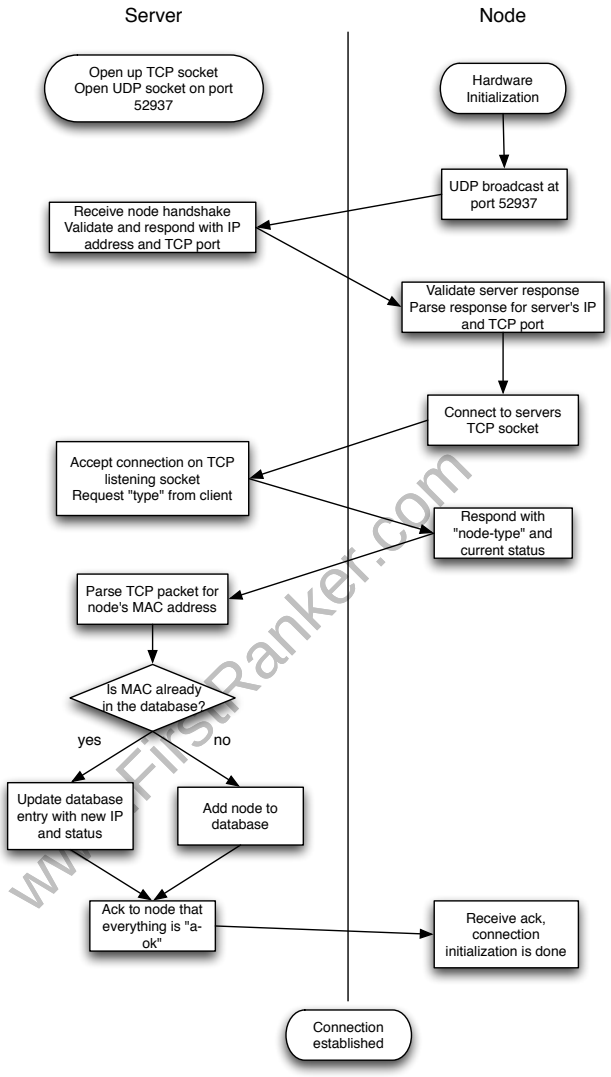


Figure 13 Node Server Connection Initialization

First, the server must perform its own setup process. This consists of opening up an available local TCP socket. The port number for this TCP socket is unknown. We did not choose a specific port since we would be performing a service broadcast from our clients anyway; this is similar to how SMB, Bonjour, etc perform service

discoveries. Once the TCP socket has finished, a UDP socket is opened and with known port of 52937. We randomly chose this number seeing as no popular services uses this port and it is not a reserved port. Since server constantly listens on both ports to accept broadcasts and socket connections and both actions block, each is on its own thread.

After hardware initialization, the outlet node performs a UDP broadcast with a destination address of *255.255.255.255* and port of 52937. This sends a datagram to every device on the network on port 52937. Most likely our server is the only device which is listening on UDP 52937. If there were other devices, it wouldn't be an issue since we perform some light handshaking. The UDP datagram consists of our secret handshake of "JKE." The server validates this handshake and responds with another key, "EKJ" as well as the IP address and TCP port of the server. This might seem like a bit of overhead, but it was the best approach when not knowing the IP address or TCP port of the server. In reality, most people don't know each port number their IP address and even then, the router's DHCP sometimes changes a device's IP address.

The node then receives the server's response, validates it, and parses the datagram for the IP address and TCP port. Now it opens a TCP connection and attempts to bind to the server's TCP socket. On the server end, the TCP socket accepts the connection, spawns a new "client decider" thread (so that other connections can still connect to the listening TCP port), and asks the node, "what type are you?" The node responds with its own type: "node-type."

The server realizes the potential client is a node and spawns a "node helper" thread. It then looks at the node's MAC address and queries the database to see if it is already a new node or a duplicate. If it is a new node, it simply adds the node with the necessary attributes to the database. If the MAC address is already in the database, the server simply updates the specific node (tuple) with new IP address, etc. Lastly, the server sends an *ack* back to the node indicating it has successfully established a connection. Meanwhile, the server adds this node connection to a list of open node

sockets. This is necessary in order to maintain an open stream between the node and server.

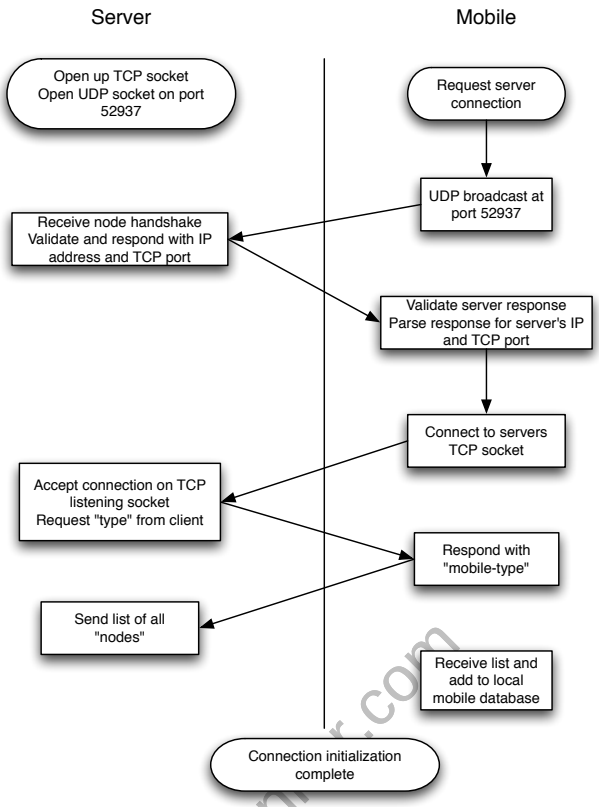


Figure 14 Server Mobile Connect Initialization

Figure 12 shows that the mobile client also performs a similar server connection process. By this point, the server has already gone through its setup process, as previously described. The mobile client broadcasts, the server responds with its own IP address and TCP port, and the mobile client then connects. When the mobile client responds to the server with the type: “mobile-type”, the server spawns a new “client helper” thread and adds this mobile client to a list of other mobile clients. Then the server queries its database and sends a list of all nodes and each of their statuses to the mobile client. This way, the mobile app is completely updated just as if it has always been connected to the system. The server-mobile client initialization is completed and now the system is ready tfor total communication.

The server does not only perform connection initialization, but it also allows for system-

wide communication: directing nodes to turn on or off and update the user interface on the mobile client. The state diagram (figure 15) describes the server's main functionality.

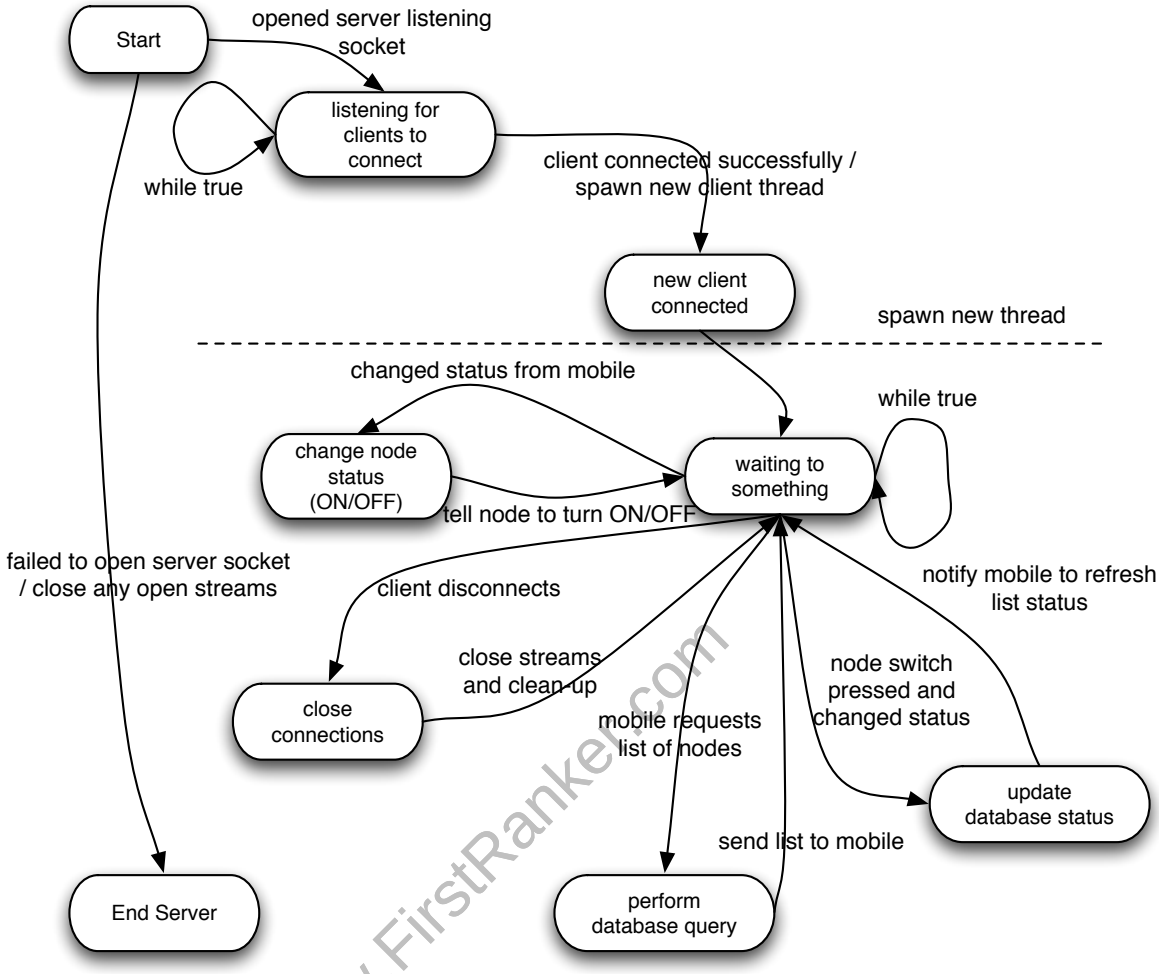


Figure 15 Server State Diagram

After a client successfully connects and the server spawns the appropriate mobile or node helper thread, the server waits for something to do. One action the server performs is notifying which node to change its status (on / off or dimmer value). This action occurs when mobile remote connected to the system has been triggered to change the status of one of the outlets from the user interface. Referring *SQLite* attributes, the server uses the *ID* of the outlet, which is the same *ID* on the mobile side and on the server node list, to send an on or off command to the appropriate node (see Packet formatting page 15). The

database is updated with the correct status of that specific node as well as the user interface for the connected mobile remotes.

Another possible action occurs when a user manually changes the status using the hardware dimmer and switch. In this case the node obviously changes its status instantly, but it also notifies the server to update the database accordingly. After, the server then notifies all connected mobile remotes to update their statuses as well.

Performing a full database query usually occurs only when a new mobile client connects; however, sometimes the mobile client doesn't display the correct node statuses and it needs to get re-synced with the server. In this case the mobile client requests a fresh new list of all nodes. The server performs a SQL query, wraps the result in a nice list, and sends it over to the mobile client who requested the list.

A rare action occurs when the mobile or node client closes its connection. Both type of clients are programed to handle this event gracefully so the server must be able to do the same. Regardless of type, the server closes any open socket streams it had with the client and it removes it from the database.

Additionally, if for some reason the server restarts and it does so too quickly, before the UDP sockets close, it will throw a `BindException` because it cannot open a socket on UDP port 52937. When this occurs, the server simply closes any other open streams and exits. Realistically, this will rarely ever happen and in that event, the user will simply just unplug the *Raspberry Pi*.

ANDROID MOBILE CLIENT

We chose Android as the platform for the mobile client because we have previous experience with the Android operating system and it allows for really simple socket

communication since it is also written in Java. We also had plenty of Android devices to develop, debug, and test our mobile remote client application.

An Android application's code runs on one main user interface (UI) thread which is perfectly okay for simple method calls. However, socket communication, database queries, and large operations can lag the UI and cause the application to freeze. To solve this issue, we created a background service, which executes socket communication and database queries on a separate thread, and a `MessageHandler` to update the UI thread so it can display the correct up-to-date node statuses. Android also supports the use of SQLite to store local application information, such as the UI persistence. There are other methods to maintain UI state; however, by synchronizing the local Android database with the main SQLite database on the server, it kept overhead overhead down and simplified much of the code. (If you need to see more information on Android, please consult the API (see sources), it is extremely helpful.)

Figure 16 demonstates the mobile application is relativiley simple: it consists of a list of outlets each with a `SeekBar` to represent the dimmer value and a ON/OFF `Switch`. Each outlet has a top main and bottom sub `TextView` to represent the naming scheme of the outlet. Under the list is the bottom half of the `ActionBar` which contains four easy-to-access buttons: add a new outlet, refresh the current list, delete the entire outlet list, and additional settings.

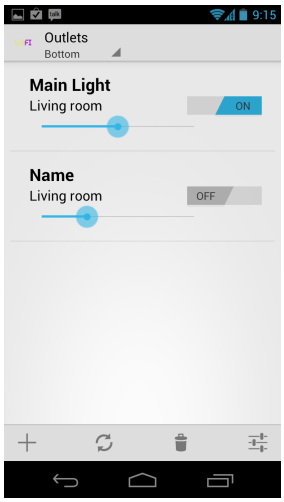


Figure 16 Screenshot of Mobile Application

The `Switch` turns the outlet node ON or OFF. When the switch is pressed, the UI thread notifies the background service to send a command to the server (see Parser) to change the node's status to the appropriate state. This allows the UI thread to be responsive to the user while using sockets to communicate with the server on a separate worker thread. The background service starts after the server-mobile connection is successfully established (see the mobile client state diagram of figure 17). As previously described, the server updates its database to reflect the change after the node's state changes.

The `SeekBar` controls with dimmer value of the outlet node. The seekbar's value goes from 0 - 240. However, the architecture of the node's dimmer requires the maximum value (completely on) to be 8 and the minimum value (completely off) to be 248. To solve this, the seekbar's value was subtracted from 248; this dimmer value was then sent to the background service. The server is notified and the dimmer value on the node is eventually changed. This event is triggered only when the user lifts his or her finger off the seekbar, not while it is moving back or forth.

The `TextViews` display the appropriate name of each outlet in the list. These names are simply for the user to identify each outlet. These names are stored in the Name attribute in the database. These names are established when the node gets added to the network and then connects to the server. (Please see the Conclusion to see how the outlet gets added as well as for the "Add" button functionality.)

The refresh button makes the local application database mirror the server database. Pressing the refresh button notifies the server that this mobile client is requesting an update on all the nodes' statuses. When the background service retrieves the list, it parses it, and then updates the local database with any changes. We essentially wipe out the local database and copy over the server's database values. The delete button completely erases local database of all outlet entities. The settings button takes the user into the applications preferences. These three buttons were mainly added for debugging and testing purposes as well as to fill in the plain user interface.

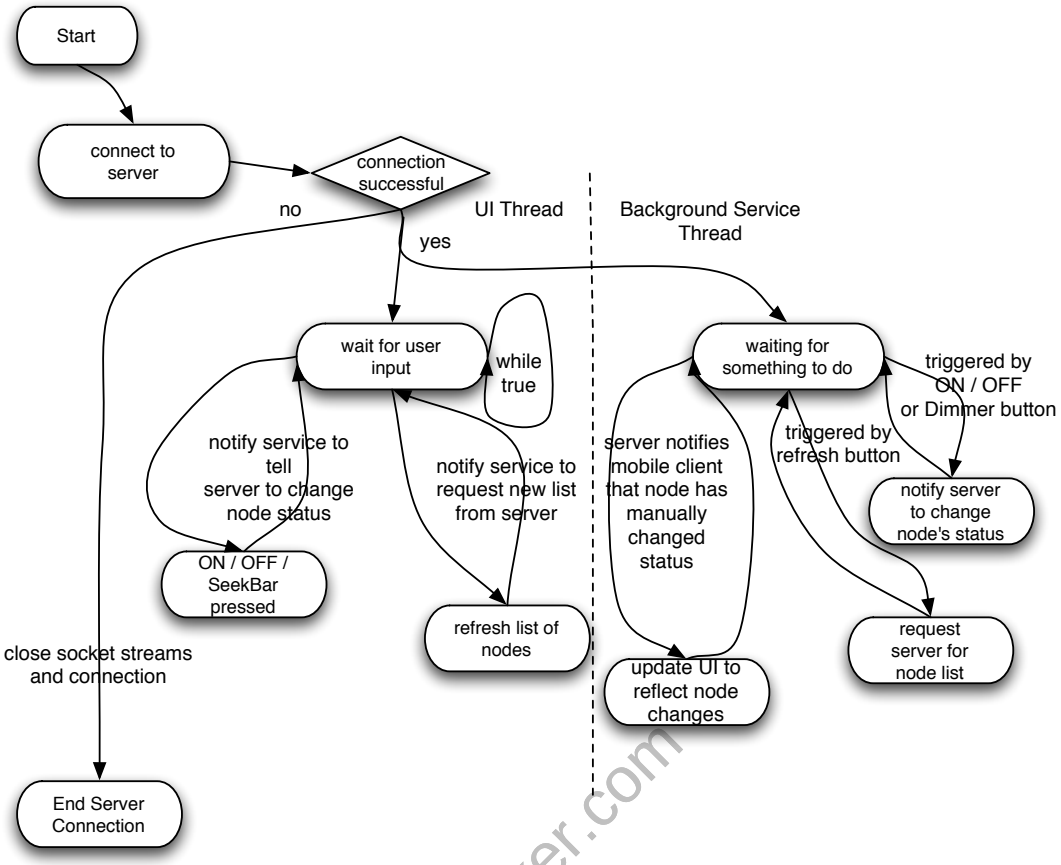


Figure 17 Mobile State Diagram

Figure 17 shows the background service also listens for socket communication *from* the server as well. This allows the mobile client to be updated on the status of any nodes which have been physically turned on or off with the dimmer switch. When this happens, the server notifies any connected mobile clients to send a refresh request to the server. These the mobile application's local database is updated to reflect these changes. A `MessageHandler` then notifies the UI thread to update itself by calling its `OnResume()` method. This forces all the widgets (`TextView`, `Switch`, `SeekBar`, etc) to be updated to their respective values. Please note that this does not happen automatically; we had to add code so that the outlets' values in the list would be pulled from the local database. This was important to do so that all mobile clients would be on the same page when looking at the user interface and the entire system would continue to operate smoothly from an end user's point of view.

The server-to-mobile socket communication uses an `Object[Input|Output]Stream`. The `Packet` class must implement the `Serializable` interface in order to be sent across the socket stream. This allows both the Android client and Java server to understand the packet as a Java object on both sides of the connection. The constructor appears as shown below:

```
public Packet () {
                mState = Action.DO_NOTHING;
                mEnable = Outlet.OFF;
                mDimVal = Outlet.MIN_DIMMER_VAL;
                mId = 0;
                mList = new ArrayList<Outlet>();
        }
```

mState is a enum which indicates the server or mobile client which action to perform. *mEnable* is an integer which indicates the outlet's state: ON or OFF. *mDimVal* is an integer which represents the outlet's dimmer status. *mId* is also an integer used to identify the outlet in the database. *mList* is the a list of outlets whose statuses are being updated through the refresh request, or changed due to a status change.

DEMO VIDEO

A demo video can be viewed on youtube.
http://www.youtube.com/watch?v=KJWV0JlShVg

CONCLUSION

We initially planned for the outlet nodes to automatically connect to the local network. The WiFi chip has an access point (AP) mode which allows a device to connect to it as a client. The Android mobile client would connect to the node, which would start in AP mode. Once connected, the application would send the SSID and passphrase to the node. The microprocessor would store these values in the EEPROM. It would also

change the boot-up mode, which is also in the EEPROM, from AP mode to client mode. The microprocessor would then reboot itself, start in client mode, read the SSID and password values in the EEPROM, and connect to the wireless network. However, this was not a key feature for our demonstration and decided to put it in the backlog.

The big emphasis on the server design and architecture was scalability and platform independence. We wanted to have the ability to reliably add several different outlets and mobile clients to our system. Also, the *Raspberry Pi* didn't have a guaranteed shipment date so we wanted to have the ability to use different hardware for the server, more specifically, a laptop. We achieved this by writing the server in Java and using a simple SQLite database. This turned out to be a good decision since the *Pi* arrived two days before the Senior Project Expo.

Finally, the embedded hardware is not limited to controlling and dimming a light; the firmware can easily be changed to enable remote control any other device, a gateway to the "internet of things."

This project allowed us to demonstrate our strengths from Electrical Engineering: PCB design, power electronics, and hardware design, Computer Engineering: hardware integration and firmware, and Computer Science: server design, database management, and mobile phone applications.

REFERENCES

[1] Android, *Android Developers SDK*, June 2012. [Online]. Available:http://developer.android.com/reference/. [Accessed Spring 2012].

[2] Atmel, *Atmega328P Datasheet*, [Online]. Available:http://www.atmel.com/Images/8271S.pdf. [Accessed January 2012]

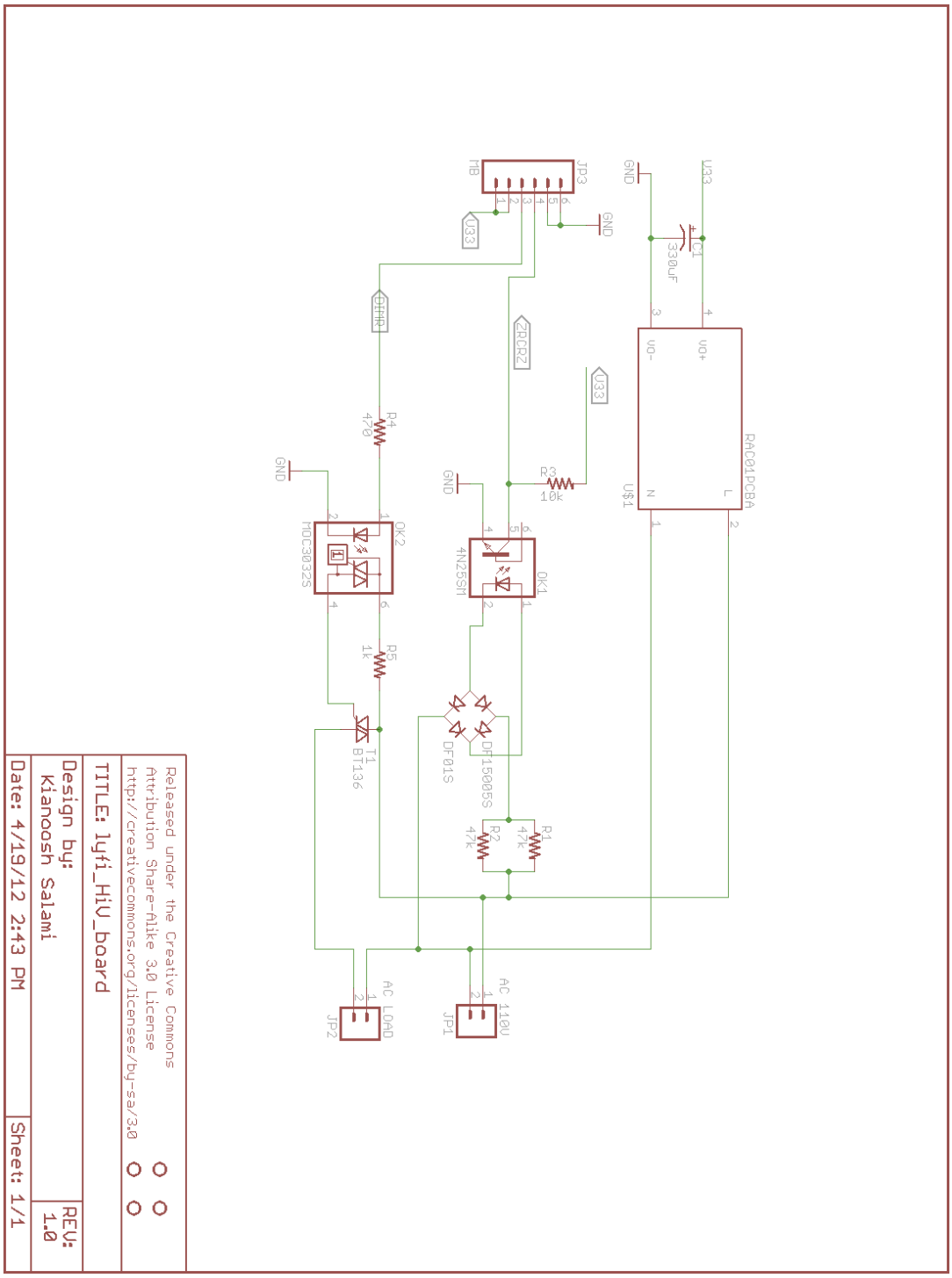
[3] FreeRTOS, *FreeRTOS API*, [Online]. Available:http://www.freertos.org/a00106.html [Accessed January 2012]

[4] Harris, Tom. *How Dimmer Switches Work,* 13 August 2002. HowStuffWorks.com [Online]. Availalbe:http://home.howstuffworks.com/dimmer-switch.htm. [Accessed: Janurary 2012]

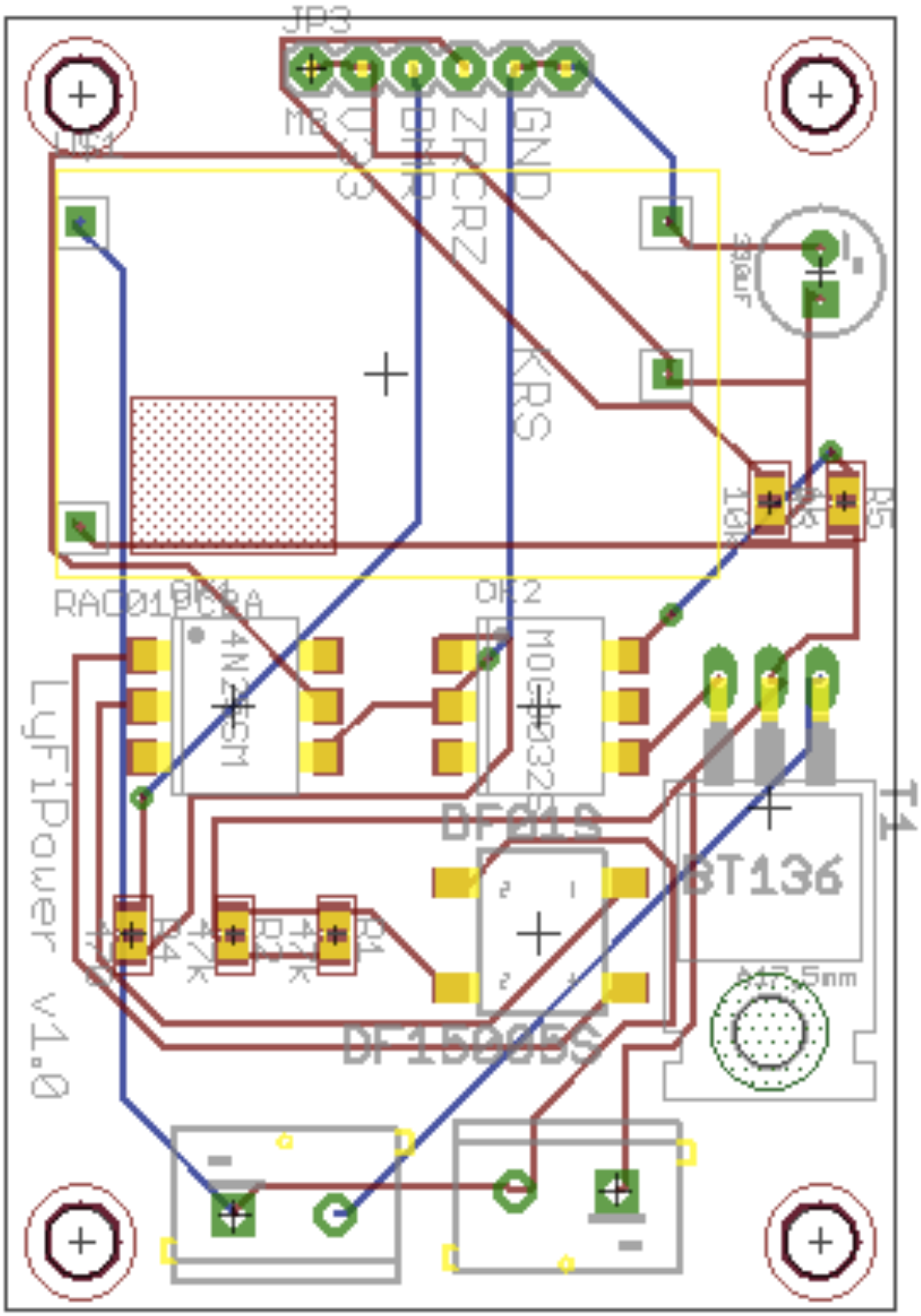
[5] Oracle, *Java Platform Standard Ed. 6*, Edition of book, 2011. [Online]. http://docs.oracle.com/javase/6/docs/api/. [Accessed Spring 2012]

APPENDIX A: HARDWARE SCHEMATIC / PCB LAYOUT

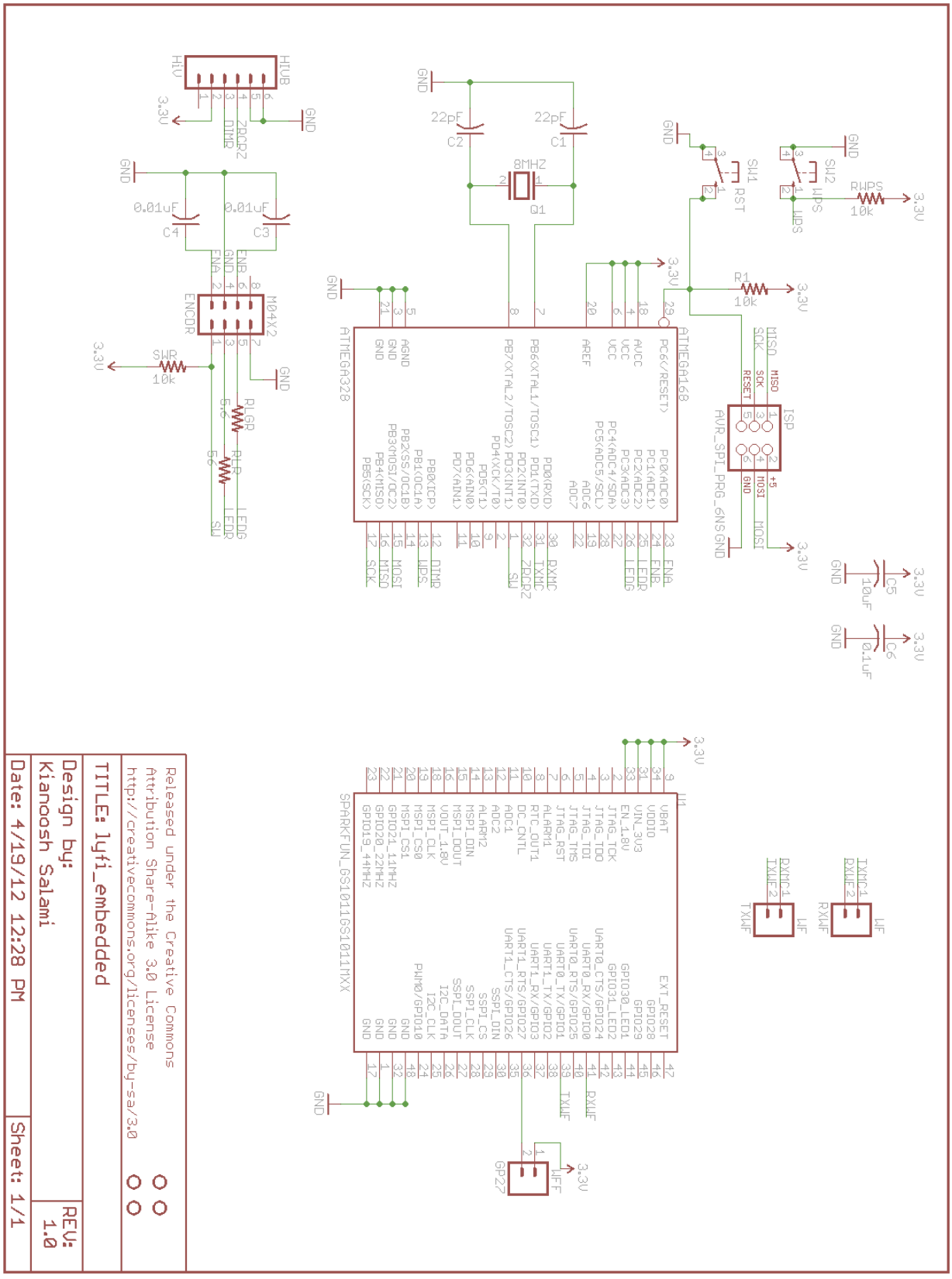
LyFi HiV Schematic: AC/DC Circuit, Zero-Cross Detector, Triac Control



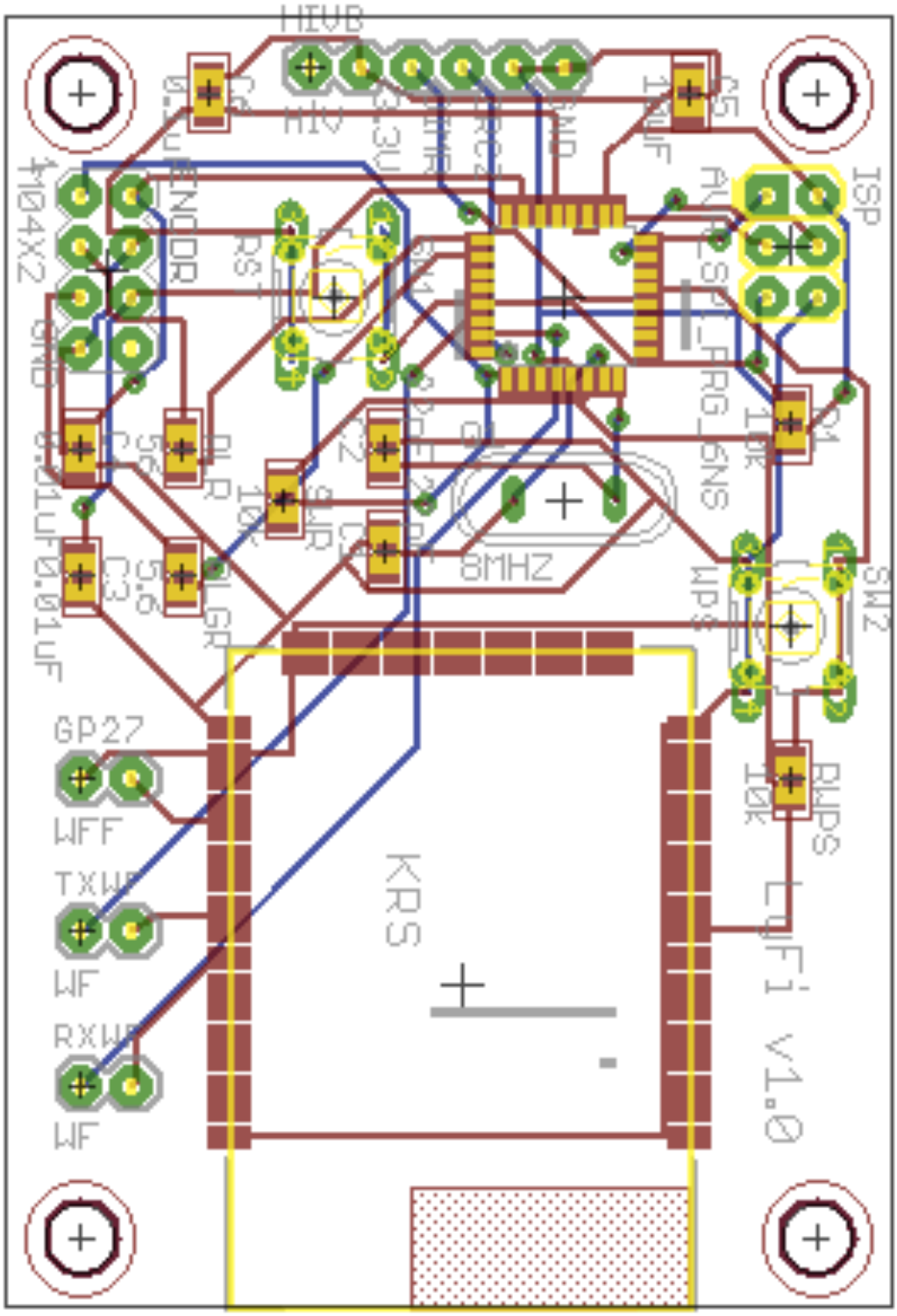
LyFi HiV PCB Layout:



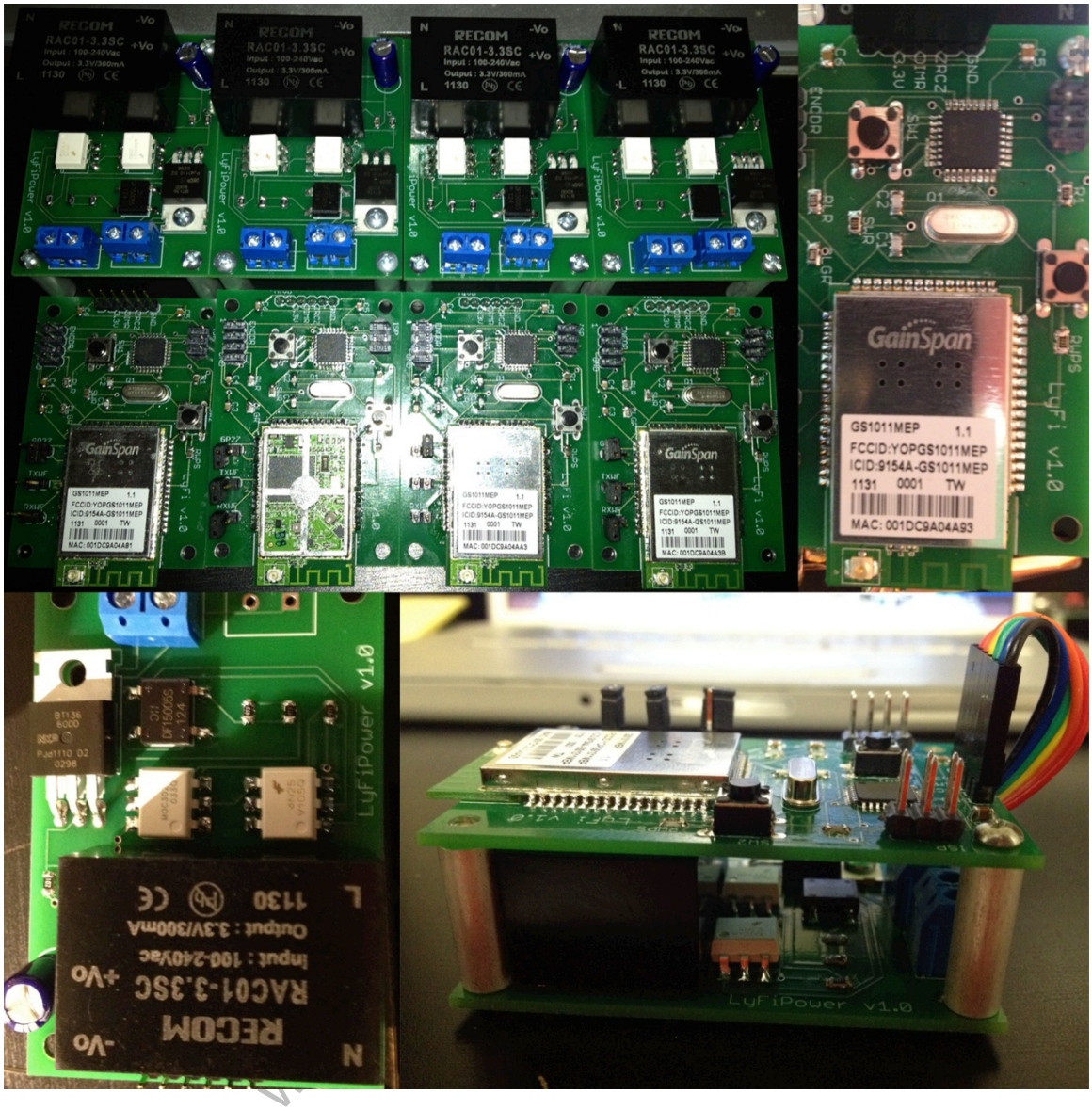
Embedded Board Schematic



Embedded Board PCB Layout



APPENDIX B: HARDWARE PHOTOS



APPENDIX C: SOURCE CODE

Firmware

```
/* FIRMWARE MAIN.C */


/*****************************************************************************
 * Header file inclusions.
 *****************************************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>

#include "FreeRTOS.h"
#include "task.h"
#include "serial.h"
#include "dimmer.h"
#include "switchDriver.h"
#include "parser.h"
#include "wifi_eeprom.h"


/*****************************************************************************
 * Private macro definitions.
 *****************************************************************************/

#define BAUD 9600
#define SER_BUF_LEN 25 //Serial buffer length

int main(void)
{
                    /*set_wf_ssid("JeebsNet", 8);
                    set_wf_pw("candy4myles316", 14);
                    set_wf_mode(1); */

                    set_wf_ssid("Enchilada", 9);
                    set_wf_pw("loadedquestion$77", 17);
                    set_wf_mode(1);


                    PORTB = 0x00;
                    if(DIMMER_EN) //Initialize dimmer specific components
                    {
                        initialize_dimmer(); //intializes dimmer
                        initialize_encoder();
                    }

                    xSerialPortInitMinimal( BAUD, SER_BUF_LEN); //Intialize serial port

                    initialize_switch();

                    //vSerialPutString( NULL, (const signed char * const)"Run", 3 );


                    /* PARSER TASK */
                    xTaskCreate( (pdTASK_CODE) vTaskParser, NULL,
                        137, (void *) NULL, 2, NULL );

                    /* SWITCH TASK ON or OFF button */
                    xTaskCreate( (pdTASK_CODE) vTaskSwitchISR, NULL,
                        125, (void *) NULL, 1, NULL );

                    /* ENCODER TASK */
                    xTaskCreate( (pdTASK_CODE) vTaskENCISR, NULL,
                        configMINIMAL_STACK_SIZE, (void *) NULL, 1, NULL );

    // Start scheduler.
    vTaskStartScheduler();

    return 0;
}
```

```
/* FreeRTOSConfig.h */

#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H


#include <avr/io.h>

/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
 * See http://www.freertos.org/a00110.html.
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION                        1
#define configUSE_IDLE_HOOK                        0
#define configUSE_TICK_HOOK                        0
#define configCPU_CLOCK_HZ                         ( ( unsigned long ) F_CPU )
#define configTICK_RATE_HZ                         ( ( portTickType ) 1000 )
#define configMAX_PRIORITIES               ( ( unsigned portBASE_TYPE ) 2 ) /*was 4 changed to 2 to
save sram */
#define configMINIMAL_STACK_SIZE   ( ( unsigned short ) 85 )
#define configTOTAL_HEAP_SIZE              ( (size_t ) ( 1500 ) )
#define configMAX_TASK_NAME_LEN            ( 1 ) /* was 8 changed to 1 because desperate for sram */
#define configUSE_TRACE_FACILITY          0
#define configUSE_16_BIT_TICKS             1
#define configIDLE_SHOULD_YIELD            1
#define configQUEUE_REGISTRY_SIZE         0
#define configUSE_MUTEXES                  1

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES              1
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet           0
#define INCLUDE_uxTaskPriorityGet          0
#define INCLUDE_vTaskDelete                                0
#define INCLUDE_vTaskCleanUpResources      0
#define INCLUDE_vTaskSuspend               0
#define INCLUDE_vTaskDelayUntil            0
#define INCLUDE_vTaskDelay                                 1


#endif /* FREERTOS_CONFIG_H */

/* switchDriver.h */
#ifndef SWITCHDRIVER_H
#define SWITCHDRIVER_H

#define DIMMER_EN 1 //1 for DIMMER 0 for RELAY
/* SEE DIMMER.H FOR RELAY PIN CONFIGURATION */

void initialize_switch();
void vTaskSwitchISR(void *pvParameters);

#endif

/* SwitchDriver.c */

#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include "FreeRTOS.h"
#include "dimmer.h"
#include "task.h"
#include "semphr.h"
#include "switchDriver.h"
#include "parser.h"

uint8_t EEMEM sw_state;

/* Enable extInterrupt INT1 */
#define INT1InterruptOn0()                \
```

```
{                                                                                \
                    EIMSK |= _BV(INT1);         \
}

#define INT1InterruptOff0()                \
{                                                                                \
                    EIMSK &= ~(_BV(INT1));                 \
}

xSemaphoreHandle switchBinSemaphor0;

void initialize_switch()
{
                    eeprom_write_byte(&sw_state, 0x01);

                    /* The rising edge of INT0 generates an interrupt request.see pg73 DS*/
                    EICRA |= ( _BV(ISC11) | _BV(ISC10) );

                    vSemaphoreCreateBinary(switchBinSemaphor0);
                    xSemaphoreTake( switchBinSemaphor0, portMAX_DELAY );

                    INT1InterruptOn0();

                    return;
}

void vTaskSwitchISR(void *pvParameters)
{
                    //static unsigned int state = 1;
                    //static unsigned int wifirun = 1;
                    /*Intialize wifi without creating a new task :) */

                    //intialize_wifi();
                    //ncloseall();
                    vTaskDelay(50 / portTICK_RATE_MS);
    connToServer();
                    vTaskDelay(50 / portTICK_RATE_MS);
    sendNodeBS();

                    for(;;)
                    {
                         /*intialize wifi */
                         /*if(wifirun)
                         {
                                  intialize_wifi();
                                  wifirun = 0;
                         }*/

                         ENC_PIN = 0x08;
                         if(xSemaphoreTake( switchBinSemaphor0, portMAX_DELAY ) == pdTRUE)
                         {
                                  INT1InterruptOff0();
                                  vTaskDelay(210 / portTICK_RATE_MS);

                                  if(eeprom_read_byte(&sw_state))
                                  {
                                           ENC_PIN = 0x08;
                                           if(DIMMER_EN)
                                           {
                                                    turnOnDimmer();
                                           }
                                           else
                                           {
                                                    turnOnRelay();
                                           }
                                           eeprom_write_byte(&sw_state, 0x00);
                                  }
                                  else
                                  {
                                           ENC_PIN = 0x04;
                                           if(DIMMER_EN)
                                           {
                                                    turnOffDimmer();
                                           }
                                           else
                                           {
                                                    turnOffRelay();
                                           }
                                           eeprom_write_byte(&sw_state, 0x01);
                                  }
                                  INT1InterruptOn0();
```

```
                        }
                    }
}


ISR(INT1_vect)
{
   static portBASE_TYPE xHigherPriorityTaskWoken;
   xHigherPriorityTaskWoken = pdFALSE;
   xSemaphoreGiveFromISR( switchBinSemaphor0, &xHigherPriorityTaskWoken );
}
/* dimmer.h */

/*
 * Dimmer driver for ATMEGA328p
 * Written By: Kianoosh Salami
 */
//

#ifndef DIMMER_H
#define DIMMER_H

/* TRIAC / RELAY CONFIGURATION */

#define TRIAC_CNTL_PORT  DDRB
#define TRIAC_CNTL_PORTO PORTB
#define TRIAC_CNTL_PIN   PB0 //TRIAC PIN OR RELAY CTRL PIN

/* ENCODER DEFINITIONS */

#define ENC_CTRL         PORTC
#define ENC_PORT         DDRC
#define ENC_PIN          PINC
#define ENC_PINMSK       0x03
#define ENC_ENA                    PC0
#define ENC_ENB                    PC1
#define ENC_LEDG          PC2
#define ENC_LEDR           PC3

#define ENCODER_POSITION_MAX  245
#define ENCODER_POSITION_MIN  8

/* DIMMER SETTINGS */

#define INITIAL_DIM 10
#define DIMMER_INC  8

/* TIMER0 SETUP */
#define tim0CTC TCCR0A = _BV(WGM01); //Normal Port Operation / CTC top OCRA
#define tim0ClkpreScale TCCR0B = _BV(CS02);  // clk prescale 256

#define TMRCMP OCR0A //TIMER COMPARE REGISTER

void initialize_dimmer();
void setDimmer(uint8_t dimmer_val);
void turnOffDimmer();
void turnOnDimmer();
void initialize_encoder();
void turnOffRelay();
void turnOnRelay();

void vTaskENCISR(void *pvParameters);

#endif /* DIMMER_H */

/* dimmer.c */

#include <avr/interrupt.h>
#include "FreeRTOS.h"
#include "dimmer.h"
#include "task.h"
#include "semphr.h"
#include "dimmer.h"
#include "serial.h"

/* Zero Cross connected to INT0 */

/* Enable extInterrupt INT0 */
#define extInterruptOn0()         \
{                                                                    \
```

```
                    EIMSK  |= _BV(INT0);       \
                    TIMSK0 |= _BV(OCIE0A);     \
}

#define extInterruptOff0()                      \
{                                                                       \
                    EIMSK &= ~(_BV(INT0));      \
                    TIMSK0 &= ~(_BV(OCIE0A));   \
}


#define PinChgInterruptOn0()                    \
{                                                                       \
                    PCICR  |= _BV(PCIE1);         \
                    PCMSK1 |= (_BV(PCINT8) | _BV(PCINT9)); \
}

#define PinChgInterruptOff0()                   \
{                                                                               \
                    PCICR &= ~(_BV(PCIE1));                             \
                    PCMSK1 &= (~(_BV(PCINT8)) & ~(_BV(PCINT9))); \
}


void initialize_dimmer()
{

                    TRIAC_CNTL_PORT |= _BV(TRIAC_CNTL_PIN);        //Enable triacpin as OUTPUT

                    /* The rising edge of INT0 generates an interrupt request.see pg73 DS*/
                    EICRA = ( _BV(ISC01) | _BV(ISC00) );
                    extInterruptOn0();

                    /* TIMER0 CONFIGURE SEE DIMMER.H */
    tim0CTC //Normal Port Operation / CTC top OCRA
    tim0ClkpreScale // clk prescale 256

                    setDimmer(INITIAL_DIM);
                    return;
}

xSemaphoreHandle encBinSemaphor0; //binary semaphor for encoder

void initialize_encoder()
{
                    ENC_PORT |= _BV(ENC_LEDR) | _BV(ENC_LEDG); //Enable LED port as OUTPUT

                    ENC_PORT &= (~(_BV(ENC_ENA)) & ~(_BV(ENC_ENB))); //Set as inputs
                    ENC_CTRL |= (_BV(ENC_ENA)) | (_BV(ENC_ENB)); //turn on pullups

                    vSemaphoreCreateBinary(encBinSemaphor0);
                    xSemaphoreTake( encBinSemaphor0, portMAX_DELAY );

                    PinChgInterruptOn0();

                    return;
}

/* Set dimmer value,
 * LOWER the number the BRIGHTER
 * HIGHER the number the MORE DIM
 * 1-255
 */
void setDimmer(uint8_t dimmer_val)
{
                    if(dimmer_val > 0 && dimmer_val < 256)
                    {
                         TMRCMP = dimmer_val;
                    }
                    return;
}

void vTaskENCISR(void *pvParameters)
{
  // enc_states[] is a fancy way to keep track of which direction
  // the encoder is turning. 2-bits of oldEncoderState are paired
  // with 2-bits of newEncoderState to create 16 possible values.
  // Each of the 16 values will produce either a CW turn (1),
  // CCW turn (-1) or no movement (0).
                    int8_t enc_states[]  = {0,-DIMMER_INC,DIMMER_INC,0,DIMMER_INC,0,0,
                    -DIMMER_INC,-DIMMER_INC,0,0,DIMMER_INC,0,DIMMER_INC,-DIMMER_INC,0};
```

```
                        static uint8_t oldEncoderState = 0;
                        uint8_t newEncoderState = 0;
                        for(;;)
                        {
                            if(xSemaphoreTake( encBinSemaphor0, portMAX_DELAY ) == pdTRUE)
                            {
                                    uint8_t tempENCVAL = TMRCMP; //Read TIMER Register
                                    PinChgInterruptOff0();
                                    vTaskDelay(7 / portTICK_RATE_MS); //Software debounce

                                    newEncoderState = ENC_PIN & ENC_PINMSK;
                                    //xSerialPutChar( serCOM1, newEncoderState, 10 );
                                    oldEncoderState <<= 2;
                                    oldEncoderState &= 0x0C; //Mask out all values except old
                                    oldEncoderState |= newEncoderState;
                                    //xSerialPutChar( serCOM1, oldEncoderState, 10 );

                                      /*-= Beacuse the lower the delay the brighter the light.*/
                                      tempENCVAL -= enc_states[oldEncoderState];

                                    if (tempENCVAL > ENCODER_POSITION_MAX)
                                    setDimmer(ENCODER_POSITION_MAX);
                                    else if (tempENCVAL < ENCODER_POSITION_MIN)
                                    setDimmer(ENCODER_POSITION_MIN);
                        else
                                            setDimmer(tempENCVAL);

                              //xSerialPutChar( serCOM1, TMRCMP, 10 ); //debug output encoder value
                                    PinChgInterruptOn0();
                            }
                        }
}

void turnOffDimmer()
{
                        extInterruptOff0();
                        TRIAC_CNTL_PORTO &= ~(_BV(TRIAC_CNTL_PIN)); //Turn off Triac

                        //ENC_PIN &= ~(_BV(ENC_LEDG));
                        //ENC_PIN |= _BV(ENC_LEDR);

}

void turnOnDimmer()
{
                        extInterruptOn0();
                        //ENC_PIN &= ~(_BV(ENC_LEDR));
                        ENC_PIN |= _BV(ENC_LEDG);
}

void turnOnRelay()
{
                        TRIAC_CNTL_PORTO |= (_BV(TRIAC_CNTL_PIN)); //Turn on Relay
}

void turnOffRelay()
{
                        TRIAC_CNTL_PORTO &= ~(_BV(TRIAC_CNTL_PIN)); //Turn off Relay
}


// This will jump when INT0 is triggered. To be used with ZERO crossing INT
ISR(INT0_vect)
{
   TRIAC_CNTL_PORTO &= ~(_BV(TRIAC_CNTL_PIN)); //Turn off Triac
   TCNT0 = 0; //Reset timer to 0
}

ISR(TIMER0_COMPA_vect) //THIS TIMER CONTROLS THE DELAY for dimmer
{
   TRIAC_CNTL_PORTO |= (_BV(TRIAC_CNTL_PIN)); //Turn on Triac
}

ISR(PCINT1_vect) //ISR for ENCODER PINS
{
                        static portBASE_TYPE xHigherPriorityTaskWoken;
                        xHigherPriorityTaskWoken = pdFALSE;
                        xSemaphoreGiveFromISR( encBinSemaphor0, &xHigherPriorityTaskWoken );
}
```

```
/* wifi_eeprom.h */

#ifndef WIFIEEPROM_H
#define WIFIEEPROM_H

/*****************************************************************************
 * EEPROM SETUP FOR WIFI SETTINGS
 *****************************************************************************/

#define MAXWFSDLEN 33
#define MAXWFPWLEN 65


void rd_wf_pw(uint8_t* wfpw);
uint8_t rd_wf_ssidlen();
uint8_t rd_wf_pwlen();
void set_wf_mode(uint8_t mode);
void set_wf_ssid(uint8_t ssid[], uint8_t len);
void set_wf_pw(uint8_t pw[], uint8_t len);
void rd_wf_ssid(uint8_t* wfssid);


#endif


/* wifi_eeprom.c */


/* WIFI_PARAM FUNCTIONS */

#include <avr/eeprom.h>

#include "wifi_eeprom.h"

uint8_t EEMEM wf_mode=0x01;

uint8_t EEMEM wf_ssidlen;
uint8_t EEMEM wf_wpalen;

uint8_t EEMEM wf_ssid[MAXWFSDLEN];
uint8_t EEMEM wf_pw[MAXWFPWLEN];


void set_wf_mode(uint8_t mode)
{
                        eeprom_write_byte(&wf_mode, mode);

                        return;
}

void set_wf_ssid(uint8_t ssid[], uint8_t len)
{
                        eeprom_write_block((const void *)ssid, (const void *)wf_ssid, len);
                        eeprom_write_byte(&wf_ssidlen, len);

                        return;
}

void set_wf_pw(uint8_t pw[], uint8_t len)
{
                        eeprom_write_block((const void *)pw, (const void *)wf_pw, len);
                        eeprom_write_byte(&wf_wpalen, len);

                        return;
}

uint8_t rd_wf_ssidlen()
{
                        return eeprom_read_byte(&wf_ssidlen);
}

uint8_t rd_wf_pwlen()
{
   return eeprom_read_byte(&wf_wpalen);
}

void rd_wf_pw(uint8_t* wfpw)
{
   uint8_t wifilen = rd_wf_pwlen();

   uint8_t wfstr[wifilen];
```

```
   eeprom_read_block((void *)wfstr, (const void *)&wf_pw, wifilen);

   //vSerialPutString( NULL, (const signed char * const)wfstr, wifilen);

   for(uint8_t i=0; i<wifilen; i++)
   {
       wfpw[i] = wfstr[i];
   }

   return;
}

void rd_wf_ssid(uint8_t* wfssid)
{
   uint8_t ssidlen = rd_wf_ssidlen();

   uint8_t wfstr[ssidlen];
   eeprom_read_block((void *)wfstr, (const void *)&wf_ssid, ssidlen);

   //vSerialPutString( NULL, (const signed char * const)wfstr, wifilen);

   for(uint8_t i=0; i<ssidlen; i++)
   {
       wfssid[i] = wfstr[i];
   }

   return;
}

/* parser.h */

#ifndef PARSER_H
#define PARSER_H


//#define WIFIDATA 4
#define EEBUFLEN 100
#define CMDDELAY 10

#define RETURN 0x0D
#define NEWLINE 0x0A
#define ESCAPE 0x1B
#define IDENTIFIER 0x24
#define ACK 0x41
#define CONN 0x43
#define STATE 0x53
#define TYPE 0x54

/*SEND STATE DEFINITIONS */
#define NONE 0
#define ATE 1
#define ATV 2
#define NDHCP 3
#define ATWWPA 4
#define ATWA 5
#define ATFNDSRV 6
#define ATWD 7
#define ATNCTCP 8
#define ATNCLOSEALL 9
#define ATBDCST 10

#define blkTime 10

uint8_t makeBuf(unsigned char);
void vTaskParser(void *pvParameters);

void connToServer();
void sendNodeBS();

void parser();
void ate(uint8_t);
void atv(uint8_t);
void atwd();
void ndhcp(uint8_t);
void wwpa();
void wa();
void nctcp();
void ncloseall();
void sendStateCheck();
void errorCheck();
uint8_t serverPacket();
```

```
void exec(uint8_t, unsigned char, uint8_t);
void encoder(uint8_t cid, char* str, uint8_t strlen);
void intialize_wifi();
void waitForResponse();
void responseReceived();
void wa_dec();
void udp_dec();
void crnl();

void broadcast();
void findserver();

#endif


/* parser.c */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>

#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
#include "serial.h"
#include "dimmer.h"
#include "switchDriver.h"
#include "parser.h"
#include "wifi_eeprom.h"


uint8_t idx = 0;
//uint8_t check = 1;
uint8_t sendState = 0;

uint8_t EEMEM eebuf[EEBUFLEN];
/* THE FOLLOWING COMMANDS ARE BLACKED OUT DUE TO AN NDA SIGNED WITH THE WIFI CHIPSET COMPANY */
uint8_t EEMEM EESERV[] = [redacted]
uint8_t EEMEM EEPKT[] = "S0$T4NODE";

uint8_t EEMEM [redacted] [] = "[redacted] ";
uint8_t EEMEM [redacted] [] = "[redacted] ";
uint8_t EEMEM [redacted] [] = "[redacted] ";
uint8_t EEMEM [redacted] [] = "[redacted]";
uint8_t EEMEM [redacted] [] = "[redacted]
uint8_t EEMEM [redacted] [] = "[redacted] ";
uint8_t EEMEM [redacted] [] = "[redacted]
uint8_t EEMEM [redacted] [] = "[redacted]
uint8_t EEMEM [redacted] [] = "[redacted] ";
uint8_t EEMEM EE255P[] = "255.255.";
uint8_t EEMEM EE255C[] = "255.255,";
uint8_t EEMEM EEPORT[] = "52937";
uint8_t EEMEM EEBRDCST[] = "S0JKE";

uint8_t EEMEM EESERVIP[21];
uint8_t EEMEM EESERVIPLEN;

xSemaphoreHandle xMutexSemaphore;
xSemaphoreHandle xSendState;


void vTaskParser(void *pvParameters)
{
   signed char recv = '0';

   xMutexSemaphore = xSemaphoreCreateMutex();
   xSendState = xSemaphoreCreateMutex();
                    xSemaphoreTake(xMutexSemaphore, portMAX_DELAY);
                    for(;;)
                    {
                    xSemaphoreGive(xMutexSemaphore);

                        xSemaphoreTake(xMutexSemaphore, portMAX_DELAY);

                        if( xSerialGetChar( NULL, &recv, 1 ) == pdTRUE )
                        {
                                makeBuf((unsigned char)recv);
                        }
                    }

}
```

```
uint8_t makeBuf(unsigned char byte)
{
   static uint8_t CRstart = 0;
   static uint8_t start = 0;
   static uint8_t CRend = 0;
   if(start == 1)
   {
      if(byte == (char)RETURN)
      {
         CRend = 1;
         return 0;
      }
      else if((CRend == 1) && (byte == (char)NEWLINE))
      {
         start = 0;
         CRend = 0;
         parser();
         idx = 0;
         return 1;
      }
      else if((CRend == 1) && (byte != (char)NEWLINE))
      {
         return 0;
      }
      else
      {
                           eeprom_write_byte((&eebuf+idx),byte);
                           idx++;

                           //xSerialPutChar(NULL,byte,10);
                           //xSerialPutChar( NULL, idx, 10 );
         return 0;
      }
   }
   else if(byte == (char)RETURN)
   {
      CRstart = 1;
      return 0;
   }
   else if((byte == (char)NEWLINE) && (CRstart == 1))
   {
      start = 1;
      CRstart = 0;
      return 0;
   }
   else
   {
      CRstart = 0;
      return 0;
   }

}

void parser()
{
   uint8_t i;
   if(eeprom_read_byte(&eebuf) == (char)ESCAPE && eeprom_read_byte(&eebuf+1) == 'S')
   {
      for(i = 2; i < idx; i++)
      {
         if(eeprom_read_byte(&eebuf+i) == (char)ESCAPE)
            if(i+1 <= idx)
               if(eeprom_read_byte(&eebuf+i+1) == 'E')
               {
                                          //xSerialPutChar(NULL,'B',10);
                  serverPacket();
               }
      }
   }
   else if(eeprom_read_byte(&eebuf) == '#')
   {
                        uint8_t atstr[idx-1];
      uint8_t attcp[9];
                        //xSerialPutChar( NULL, idx, 10 );

     // eeprom_write_byte(&EESERVIPLEN, (idx-1));


                        for(uint8_t i=1; i<idx; i++)
                        {
```

```
                        atstr[i-1] = eeprom_read_byte(&eebuf+i);
      }

                      //vSerialPutString(NULL, (const signed char * const)atstr,(idx-1));
      eeprom_read_block((void *)attcp, (const void *)&EEatnctcp, 9);

      crnl();
      vSerialPutString( NULL, (const signed char * const)attcp, 9);
                      vSerialPutString( NULL, (const signed char * const)atstr, idx-1);
                      crnl();

                      sendStateCheck();
   }

   return;
}

void ate(uint8_t echo)
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATE;
   taskEXIT_CRITICAL();

   uint8_t atstr[3];

   eeprom_read_block((void *)atstr, (const void *)&EEate, 3);

   crnl();
   vSerialPutString( NULL, (const signed char * const)atstr, 3);
   xSerialPutChar( NULL, (echo + '0'), blkTime );
   crnl();

   waitForResponse();

   return;
}

void atwd()
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATWD;
   taskEXIT_CRITICAL();

   uint8_t atstr[5];

   eeprom_read_block((void *)atstr, (const void *)&EEatwd, 5);

   crnl();
   vSerialPutString( NULL, (const signed char * const)atstr, 5);
   crnl();

   waitForResponse();

   return;

}

void atv(uint8_t verbose)
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATV;
   taskEXIT_CRITICAL();

   uint8_t atstr[3];

   eeprom_read_block((void *)atstr, (const void *)&EEatv, 3);

   crnl();
   vSerialPutString( NULL, (const signed char * const)atstr, 3);
   xSerialPutChar( NULL, (verbose + '0'), blkTime );
   crnl();

   waitForResponse();

   return;
```

```
}

void ndhcp(uint8_t mode)
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = NDHCP;
   taskEXIT_CRITICAL();

   uint8_t atstr[9];

   eeprom_read_block((void *)atstr, (const void *)&EEatndhcp, 9);

   crnl();
   vSerialPutString( NULL, (const signed char * const)atstr, 9);
   xSerialPutChar( NULL, (mode + '0'), blkTime );
   crnl();

   waitForResponse();

   return;
}

void wwpa()
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   uint8_t wifilen = rd_wf_pwlen();

   uint8_t wfstr[wifilen];

   rd_wf_pw(wfstr);

   taskENTER_CRITICAL();
   sendState = ATWWPA;
   taskEXIT_CRITICAL();

   uint8_t atstr[8];
   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   eeprom_read_block((void *)atstr, (const void *)&EEatwwpa, 8);

   crnl();
   vSerialPutString( NULL, (const signed char * const)atstr, 8);
   vSerialPutString(NULL, (const signed char * const)wfstr, wifilen);
   crnl();

   waitForResponse();

   return;

}

void wa()
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   uint8_t ssidlen = rd_wf_ssidlen();

   uint8_t wfstr[ssidlen];

   uint8_t atstr[6];

   rd_wf_ssid(wfstr);

   taskENTER_CRITICAL();
   sendState = ATWA;
   taskEXIT_CRITICAL();

   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   eeprom_read_block((void *)atstr, (const void *)&EEatwa, 6);

   crnl();
   vSerialPutString( NULL, (const signed char * const)atstr, 6);
   vSerialPutString(NULL, (const signed char * const)wfstr, ssidlen);
   crnl();

   waitForResponse();

   return;
}
```

```
/*
void nctcp()
{
   //while(sendState != 0)
   taskENTER_CRITICAL();
   sendState = ATNCTCP;
   taskEXIT_CRITICAL();

   char str[WIFIDATA] = "XXat+ntcp=";
   str[0] = RETURN;
   str[1] = NEWLINE;
}
*/

void ncloseall()
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATNCLOSEALL;
   taskEXIT_CRITICAL();

   uint8_t wfstr[12];
   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   eeprom_read_block((void *)wfstr, (const void *)&EEatcloseall, 12);

   crnl();
   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   vSerialPutString( NULL, (const signed char * const)wfstr, 12);
   crnl();

   waitForResponse();

   return;
}

void findserver()
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATFNDSRV;
   taskEXIT_CRITICAL();

   uint8_t atstr[9];

   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   eeprom_read_block((void *)atstr, (const void *)&EEatncudp, 9);
   vTaskDelay(CMDDELAY / portTICK_RATE_MS);

   crnl();
   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   vSerialPutString( NULL, (const signed char * const)atstr, 9);

   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   eeprom_read_block((void *)atstr, (const void *)&EE255P, 8);
   vSerialPutString( NULL, (const signed char * const)atstr, 8);

   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   eeprom_read_block((void *)atstr, (const void *)&EE255C, 8);
   vSerialPutString( NULL, (const signed char * const)atstr, 8);


   vTaskDelay(CMDDELAY / portTICK_RATE_MS);
   eeprom_read_block((void *)atstr, (const void *)&EEPORT, 5);
   vSerialPutString( NULL, (const signed char * const)atstr, 5);
   crnl();

   waitForResponse();

   return;

}

void broadcast()
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATBDCST;
   taskEXIT_CRITICAL();
```

```
   uint8_t atstr[9];

   crnl();
   xSerialPutChar( NULL, ESCAPE, blkTime );

   eeprom_read_block((void *)atstr, (const void *)&EEBRDCST, 5);
   vSerialPutString( NULL, (const signed char * const)atstr, 5);


   xSerialPutChar( NULL, ESCAPE, blkTime );
   xSerialPutChar( NULL, 'E', blkTime );

   waitForResponse();

   return;
}

void connToServer()
{
   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATBDCST;
   taskEXIT_CRITICAL();

   uint8_t atstr[18];

   crnl();

   eeprom_read_block((void *)atstr, (const void *)&EEatnctcp, 9);
   vSerialPutString( NULL, (const signed char * const)atstr, 9);


   eeprom_read_block((void *)atstr, (const void *)&EESERV, 18);
   vSerialPutString( NULL, (const signed char * const)atstr, 18);

   crnl();

   waitForResponse();

   return;
}

void sendNodeBS()
{

   xSemaphoreTake(xSendState, portMAX_DELAY);

   taskENTER_CRITICAL();
   sendState = ATBDCST;
   taskEXIT_CRITICAL();

   uint8_t atstr[9];

   crnl();
   xSerialPutChar( NULL, ESCAPE, blkTime );
   eeprom_read_block((void *)atstr, (const void *)&EEPKT, 9);
   vSerialPutString( NULL, (const signed char * const)atstr, 9);
   xSerialPutChar( NULL, ESCAPE, blkTime );
   xSerialPutChar( NULL, 'E', blkTime );

   waitForResponse();

   return;

}



void sendStateCheck()
{
   switch(sendState)
   {
      case ATE:
                        errorCheck();
                        break;
      case ATV:
         errorCheck();
         break;
      case NDHCP:
```

```
         errorCheck();
         break;
      case ATWWPA:
         errorCheck();
         break;
      case ATWA:
         wa_dec();
         break;
      case ATWD:
                           errorCheck();
      case ATFNDSRV:
                           udp_dec();
         break;
      case ATNCTCP:
         break;
      case ATNCLOSEALL:
         errorCheck();
         break;

   }

   return;
}

void errorCheck()
{
   uint8_t val =  eeprom_read_byte(&eebuf);
   if(val == '0' || val == 'O')
   {
//     check = 0;
                        taskENTER_CRITICAL();
     sendState = 0;
     taskEXIT_CRITICAL();
                        responseReceived();

     // check = 0;
   }
   else
   {
      //check = 1;
                        // waitForResponse();
   }

   return;
}

void wa_dec()
{
   if(eeprom_read_byte(&eebuf) == 0x0A)//0x20)
   {
     //check = 0;
                        taskENTER_CRITICAL();
     sendState = 0;
     taskEXIT_CRITICAL();
                        responseReceived();
   }
   else
   {
      //check = 1;
                        // waitForResponse();
   }

   return;
}

void udp_dec()
{
   if(eeprom_read_byte(&eebuf) == '7')
   {
     //check = 0;
                        taskENTER_CRITICAL();
     sendState = 0;
     taskEXIT_CRITICAL();
                        //responseReceived();
   }
   else
   {
      //check = 1;
                       //  waitForResponse();
   }
```

```
   return;
}

uint8_t serverPacket()
{
   uint8_t len, cid;
   unsigned char tempCID;
   unsigned char tempLen;
   unsigned char servAction = 0;
   if(idx > 5)
   {
      tempCID = (char)eeprom_read_byte(&eebuf+2);

      if(eeprom_read_byte(&eebuf+3) == (char)IDENTIFIER)
      {
         switch(eeprom_read_byte(&eebuf+4))
         {
            case (char)ACK:
               servAction = (char)ACK;
               break;
            case (char)CONN:
               servAction = (char)CONN;
               break;
            case (char)STATE:
               servAction = (char)STATE;
               break;
            case (char)TYPE:
               servAction = (char)TYPE;
               break;
         }
         tempLen = (char)eeprom_read_byte(&eebuf+5);
         len = tempLen - '0';

         cid = tempCID - '0';
         exec(cid, servAction, len);
         return 1;
      }
   }
   return 0;
}

void exec(uint8_t cid, unsigned char action, uint8_t len)
{
                     switch(action)
                     {
                          case (char)ACK:
                                   if(eeprom_read_byte(&eebuf+6) == '1')
                                   {
                                            if(DIMMER_EN)
                                            {
                                                     turnOnDimmer();
                                            }
                                            else
                                            {
                                                     turnOnRelay();
                                            }
                                   }
                                   else if(eeprom_read_byte(&eebuf+6) == '0')
                                   {
                                            if(DIMMER_EN)
                                            {
                                                     turnOffDimmer();
                                            }
                                            else
                                            {
                                                     turnOffRelay();
                                            }
                                   }
                                setDimmer((unsigned int)eeprom_read_byte(&eebuf+7));
            break;
          case (char)CONN:

            break;
          case (char)STATE:

            break;
          case (char)TYPE:
            //encoder(1, "$T4NODE", 7);
            break;
                     }
                     vSerialPutString( NULL, (const signed char * const)"EXEC", 4 );
```

```
                    return;
}


void encoder(uint8_t cid, char* str, uint8_t strlen)
{

   xSerialPutChar( NULL, ESCAPE, blkTime );
   xSerialPutChar( NULL, 'S', blkTime );
   xSerialPutChar( NULL, (cid + '0'), blkTime );
   vSerialPutString(NULL, (const signed char * const)str, strlen);
   xSerialPutChar( NULL, ESCAPE, blkTime );
   xSerialPutChar( NULL, 'E', blkTime );
   xSerialPutChar( NULL, RETURN, blkTime );

   return;
}

void intialize_wifi()
{

                    crnl();
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
                    ate(0);
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
                    atv(0);
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
                    atwd();
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
                    ndhcp(1);
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
    wwpa();
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
    wa();
    //vTaskDelay(CMDDELAY / portTICK_RATE_MS);
                    //ncloseall();
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
                    findserver();
    vTaskDelay(CMDDELAY / portTICK_RATE_MS);
                    broadcast();

                    //waitForResponse();

                    return;
}

void waitForResponse()
{
                    xSemaphoreGive(xMutexSemaphore);

                    return;
}

void responseReceived()
{
                    xSemaphoreGive(xSendState);

                    return;
}

void crnl()
{
                    xSerialPutChar( NULL, RETURN, blkTime );
                    xSerialPutChar( NULL, NEWLINE, blkTime );

                    return;
}
```

SERVER CODE

```
////////////////////////////////////////////////////////////////////////
//////////                Server Code                         ////////////
////////////////////////////////////////////////////////////////////////


/*
* ServerMain.java - class which has the main method to start the server
*/
package sp.server;


import java.io.IOException;


public class ServerMain {


        public static void main(String[] args) throws IOException {
                Server server = new Server(); //opens up available TCP port and startsUDP broadcast
service
                //start listening for TCP/IP connections
                while (true)
                        server.startSocketListener();
        }
}




/*
* Server.java - main class which is repsponsible the server architecture
*/


package sp.server;


import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.nio.charset.Charset;
import java.util.HashMap;
import java.util.Map;



public class Server {


        //static actions
        private final static byte DEFAULT =  '%';


        //constants
        private final static String handshakeRecv = "JKE";
        private final static String handshakeResp = "EKJ";
        private final static int TCP_PORT_NUM = 53971;
        private final static int SERVICE_PORT_NUM = 52937;
        private final static int INVALID_PORT = -33;
        private final static String ASCII = "US-ASCII";
        private final static boolean DEBUG = true;
        //instance fields
        private static int kServerPort = INVALID_PORT;
        private ServerSocket kServerSocket = null;
        public static Map<Integer, Node> kNodeMap;
        private int kId = 0;
```

```
	//Constructor, starts broadcast service thread after opening TCP listening socket
	public Server () {
		//Create serverSocket
		try {
			//create listening TCP/IP socket on any open port on the local host
			kServerSocket = new ServerSocket(TCP_PORT_NUM, 0,
InetAddress.getLocalHost());
		} catch (IOException ex) {
			System.err.println("Could not listen on default port");
			ex.printStackTrace();
			closeListeningSocket();
			System.exit(-3);
		}
		kServerPort = kServerSocket.getLocalPort();
		System.out.println("Host " + kServerSocket.getInetAddress().getHostAddress()
			+ " is listening for TCP connection on port " +
kServerSocket.getLocalPort());


		//Start UDP broadcast service thread
		ServiceThread service = new ServiceThread();
		new Thread(service).start();


		//initialize Map
		kNodeMap = new HashMap<Integer, Node>();
	}


	/*
	* This starts the socket for receiving TCP/IP connections from clients.
	* This then spawns threads per connected client the server accepts.
	*/
	public void startSocketListener() {


		Socket clientSocket = null;
		ClientDecider unknownClient = null;


		try {
			clientSocket = kServerSocket.accept(); //blocks
			unknownClient = new ClientDecider(clientSocket);
			new Thread(unknownClient).start();
		} catch (IOException ex) {
			ex.printStackTrace();
			System.err.println("Accept failed");
			closeListeningSocket();
			System.exit(-3);
		}
	}


	/*
	* Close the ServerSocketListener (the one acception TCP clients)
	*/
	private void closeListeningSocket() {
		if (null != kServerSocket) {
			try {
				kServerSocket.close();
			} catch (IOException e) {
				if(DEBUG)
					System.err.println("Error closing tcpListeningSocket");
			}
		}
	}


	private boolean contentEquals(byte[] arg1, byte[] arg2, int len) {


		boolean result = true;
		if (len <= 0) {
			return false;
		}
		for (int i = 0; i < len; i++) {
			if(arg1[i] != arg2[i]) {
				result = false;
				break;
			}
```

```
                }
                return result;
        }

        private class ClientDecider implements Runnable {

                private Socket clientSocket;

                public ClientDecider(Socket clientSocket) {
                        this.clientSocket = clientSocket;
                }

                @       Override
                public void run() {
                        System.out.println("Connected with client: " +
clientSocket.getInetAddress().getCanonicalHostName());
                        BufferedInputStream inStream = null;
                        //ObjectInputStream inStream = null;
                        //BufferedReader inStream = null;
                        BufferedOutputStream outStream = null;

                        try {
                                //inStream = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                                inStream = new BufferedInputStream(clientSocket.getInputStream());
                                //inStream = new ObjectInputStream(clientSocket.getInputStream());
                                outStream = new
BufferedOutputStream(clientSocket.getOutputStream());

                                //what type is the client?
                                Message mess = new Message(Message.ACTION_TYPE, null, (byte)0);
                                /*outStream.write(mess.toByteDataArray());
                                outStream.flush();
                                */



                                byte[] buff = new byte[128];
                                System.out.println("waiting to read");
                                int numRead = inStream.read(buff);
                                mess = Message.byteToMessage(buff, numRead);
                                System.out.println("test " + mess);
                                //if the UnknownClient is NODETYPE, do nodetype stuff
                                if (mess.dataBufferAsString().equals(Message.NODE_TYPE)) {
                                        System.out.println(mess.dataBufferAsString());

        /*dbUtil.checkNodeInDatabase(getHardwareAddress(clientSocket),

          clientSocket.getInetAddress().getHostAddress(),

          "status");
                                 */
                                        Outlet outlet = new Outlet(kId++,
                                                        Outlet.TYPE_DIMMER,
                                                        "Main Light",
                                                        Outlet.OFF,
                                                        Outlet.MIN_DIMMER_VAL,

        clientSocket.getInetAddress().getHostAddress(),
                                                        getHardwareAddress(clientSocket));


                                        Node node = new Node(clientSocket, inStream, outStream,
outlet);
                                        //add to map
                                        System.out.println("added node to map with id of: " +
outlet.getkId());
                                        kNodeMap.put(Integer.valueOf(node.getkOutlet().getkId()),
node);


                                        NodeClientHelper nodeClient = new NodeClientHelper(node);
                                        new Thread(nodeClient).start();
                                        //streams get closed in finally
```

```
                                } else { //its a mobileApp

                                        MobileClientHelper mobileClient = new
MobileClientHelper(clientSocket);
                                        new Thread(mobileClient).start();


                                }

                        } catch (IOException e) {


                                System.out.println("Closing \"decider\" streams only");
                                if (null != inStream) {
                                        try {
                                                inStream.close();
                                        } catch (IOException ex) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                        }
                                }
                                if (null != outStream) {
                                        try {
                                                outStream.close();
                                        } catch (IOException ex1) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                        }
                                }
                        }

                }
        }


        private class NodeClientHelper implements Runnable {


                private Node kNode;



                public NodeClientHelper(Node node) {
                        kNode = node;
                }


                @Override
                public void run() {
                        System.out.print("NodeClient running");
                        System.out.println(" ID: " + kNode.getkOutlet().getkId());
                        while (true) {
                                try {
                                        Thread.sleep(50);
                                } catch (InterruptedException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }
                                int dimVal = 0;
                                if (kNode.isStateChanged()) {
                                        System.out.println("My state: " +
kNode.getkOutlet().getkStatus());
                                        switch (kNode.getkOutlet().getkStatus()) {
                                        case Outlet.ON:
                                                System.out.println("ID: " +
kNode.getkOutlet().getkId() + "Turning light on now.....");
                                                dimVal = kNode.getkOutlet().getkDimmer();
                                                System.out.println("dimval: " + dimVal);
                                                byte[] data = new byte[2];
                                                data[0] = '1';
                                                data[1] = (byte)(Math.min((dimVal & 0xFF), 255));
                                                Message mess = new Message('A', data , (byte)2);
                                                try {
```

```
kNode.getkOutStream().write(mess.toByteDataArrayHack());
                                                        kNode.getkOutStream().flush();
                                                } catch (IOException e) {
                                                        //TODO
                                                        e.printStackTrace();
                                                }
                                                kNode.setStateChanged(false);
                                                break;


                                        case Outlet.OFF:
                                                System.out.println("ID: " +
kNode.getkOutlet().getkId() + "Turning light off now.....");
                                                dimVal = kNode.getkOutlet().getkDimmer();
                                                System.out.println("dimval: " + dimVal);
                                                byte[] data1 = new byte[2];
                                                data1[0] = '0';
                                                data1[1] = (byte)(Math.min((dimVal & 0xFF), 255));
                                                Message mess1 = new Message('A', data1 , (byte)2);
                                                try {

kNode.getkOutStream().write(mess1.toByteDataArrayHack());
                                                        kNode.getkOutStream().flush();
                                                } catch (IOException e) {
                                                        // TODO Auto-generated catch block
                                                        e.printStackTrace();
                                                }
                                                kNode.setStateChanged(false);
                                                break;


                                        default:
                                                System.out.println("Kian's dream....default");
                                                break;
                                        }



                                }
                        }
                }

        }

        private class MobileClientHelper implements Runnable {

                private Socket clientSocket;



                public MobileClientHelper(Socket clientSocket) {
                        this.clientSocket = clientSocket;
                }

                @Override
                public void run() {
                        System.out.println("Connect with client: " +
clientSocket.getInetAddress().getCanonicalHostName());

                        ObjectInputStream inStream = null;
                        ObjectOutputStream outStream = null;

                        ServerProtocol protocol = new ServerProtocol();
                        Packet recvPacket = null;

                        try {
                                inStream = new ObjectInputStream(clientSocket.getInputStream());
                                outStream = new ObjectOutputStream(clientSocket.getOutputStream());

                                System.out.println("Finished streams");
                                while (true) {
                                        recvPacket = (Packet)inStream.readObject();
```

```
                                protocol.process(outStream, recvPacket);
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                } catch (ClassNotFoundException e) {
                        System.err.println("Ohoh, class not found!");
                        e.printStackTrace();
                } finally {


                        System.out.println("Closing sockets/streams in ClientSokcet");
                        if (null != inStream) {
                                try {
                                        inStream.close();
                                } catch (IOException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }
                        }
                        if (null != outStream) {
                                try {
                                        outStream.close();
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                        if (null != clientSocket) {
                                try {
                                        clientSocket.close();
                                } catch (IOException ex) {
                                        ex.printStackTrace();
                                }
                        }
                }


        }


}


/*
* Helper method to get the MAC address from a Socket
*/
private String getHardwareAddress(Socket socket) {
        String result = null;
        try {
  NetworkInterface ni = NetworkInterface.getByInetAddress(socket.getInetAddress());

  if (ni != null) {
      byte[] mac = ni.getHardwareAddress();
      if (mac != null) {
          /*
           * Extract each array of mac address and convert it
           * to hexa with the following format
           * 08-00-27-DC-4A-9E.
           */
        result = "";
          for (int i = 0; i < mac.length; i++) {
              result += String.format("%02X%s",
                      mac[i], (i < mac.length - 1) ? "-" : "");
          }
      } else {
          System.out.println("Address doesn't exist or is not " +
                  "accessible.");
      }
  } else {
      System.out.println("Network Interface for the specified " +
              "address is not found.");
  }


} catch (SocketException e) {
   e.printStackTrace();
}
        return result;
}
```

```
	/*
	* This opens a UDP port at SERVICE_PORT_NUM and accepts broadcasts from potential clients.
	* It checks the handshake and responds to the client with the correct InetAddress(IP) and
	* port of the TCP/IP socket.
	* This basically tells the client which IP and Port to attempt to connect.
	*/
	private class ServiceThread implements Runnable {


		private void startServiceSocket() {
			DatagramSocket service = null;
			try {
				while (kServerPort == INVALID_PORT) {
					if (DEBUG)
						System.out.println("Waiting");
				}
				service = new DatagramSocket(SERVICE_PORT_NUM);
				byte[] buff = new byte[256];
				DatagramPacket datagram = new DatagramPacket(buff, buff.length);
				System.out.println("Receiving broadcasts on port " +
service.getLocalPort());


				while (true) {
					service.receive(datagram); //blocks until broadcast is
received
					System.out.println("Client broadcasted");
					//get datagram packet and get handshake from client's
broadcast
					String str = new String(datagram.getData(), 0,
datagram.getLength(), Charset.forName(ASCII));
					if (DEBUG)
						System.out.println("recv: " + str);
					//validate string
					str = validateHandshake(str);
					if (null == str) {
						continue;
					}
					if (DEBUG)
						System.out.println("response: " + str);


					//get IP and port of client to respond to
					InetAddress addr = datagram.getAddress();
					int port = datagram.getPort();
					if (DEBUG) {
						System.out.println(addr.getHostAddress());
						System.out.println(port);
					}
					for (int i = 2; i < 3; i++) {
						buff = str.getBytes(Charset.forName(ASCII));
						//construct new datagram to respond to client with
buff + ip + port
						datagram = new DatagramPacket(buff, buff.length,
InetAddress.getByName(addr.getHostAddress()), port);
						service.send(datagram);
						System.out.println("Responded to client at" +
addr.getHostAddress());
						try {
							Thread.sleep(3000);
						} catch (InterruptedException e) {
							// TODO Auto-generated catch block
							e.printStackTrace();
						}
					}
				}


			} catch (IOException ex){
				ex.printStackTrace();
				if (null != service)
					service.close();
				System.exit(-3);
			}
		}


		/*
		* This validates the received handshake in the buffer.
```

```
		* If its correct, it will append appropriate response and add port number.
		*/
		private String validateHandshake(String str) {
			if (null == str || !str.equals(handshakeRecv)) {
				if (DEBUG)
					System.out.println("Invalid recv string: str");
				return null;
			}
			else {
				StringBuilder builder = new StringBuilder();
				builder.append(handshakeRecv);
				builder.append(handshakeResp);
				builder.append("@");
				builder.append(kServerPort);
				return builder.toString();
			}
		}


		@Override
		public void run() {
			startServiceSocket();
		}
	}

}



/*
* ServerProtocol.java - handles communication between mobile and server
*/
package sp.server;


import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;


import sp.server.Packet.Action;


public class ServerProtocol {



	public int process(ObjectOutputStream out, Packet recPacket) throws IOException {
		System.out.println("Processing streams");
		System.out.println(recPacket.getmDimVal());
		System.out.println(recPacket.getmState().toString());
		Packet.Action state = recPacket.getmState();
		Packet packet = null;



		switch (state) {


		case PERFORM_QUERY:
			packet = new Packet();
			packet.setmState(Action.PERFORM_QUERY);
			Iterator<Entry<Integer, Node>> iter = Server.kNodeMap.entrySet().iterator();
			int i = 0;
			while(iter.hasNext()) {
				Map.Entry<Integer, Node> mEntry = (Map.Entry<Integer,
Node>)iter.next();
				mEntry.getKey().intValue();
				i++;
				packet.getmList().add(mEntry.getKey().intValue(),
mEntry.getValue().getkOutlet());
			}
			System.out.println("Number of shits: " + i);
			out.writeObject(packet);
			//out.flush();
			System.out.println("Sent list of objects");
```

```
                        break;


                case TURN_ON_FROM_APP:
                        System.out.println("TURN ON");
                        //Node node =
Server.kNodeMap.get(Integer.valueOf(recPacket.getmList().get(0).getkId()));
                        Node node = Server.kNodeMap.get(Integer.valueOf(recPacket.getmId()));
                        System.out.println(node.toString());
                        node.getkOutlet().setkDimmer(recPacket.getmDimVal());
                        node.getkOutlet().setkStatus(Outlet.ON);
                        node.setStateChanged(true);
                        break;


                case TURN_OFF_FROM_APP:
                        System.out.println("TURN OFF");
                        //Node node1 =
Server.kNodeMap.get(Integer.valueOf(recPacket.getmList().get(0).getkId()));
                        Node node1 = Server.kNodeMap.get(Integer.valueOf(recPacket.getmId()));
                        System.out.println(node1.toString());
                        node1.getkOutlet().setkDimmer(recPacket.getmDimVal());
                        node1.getkOutlet().setkStatus(Outlet.OFF);
                        node1.setStateChanged(true);
                        System.out.println(node1.toString());
                        break;


                case DO_NOTHING:
                        break;


                case SERVER_ACK:
                        System.out.println("ACK");
                        break;


                default:
                        break;
                }
                return 0;
        }
}



/*
* Outlet.java - java object to represent an outlet, used in server and mobile
*/
package sp.server;


import java.io.Serializable;


public class Outlet implements Serializable {
        //constants
        public static final int OFF = 0;
        public static final int ON = 1;
        public static final int DEFAULT_ID = 8385;
        public static final String TYPE_DEFAULT = "default";
        public static final String TYPE_DIMMER = "dimmer";
        public static final String TYPE_RELAY = "relay";
        public static final int MAX_DIMMER_VAL = 8;
        public static final int MIN_DIMMER_VAL = 248;


        private int kId;
        private String kType;
        private String kName;
        private int kStatus;
        private int kDimmer;
        private String kIPaddress;
        private String kMacAddress;


        public Outlet () {
                kId = DEFAULT_ID;
                kType = TYPE_DEFAULT;
                kName = null;
                kStatus = OFF;
```

```
                kDimmer = MIN_DIMMER_VAL;
                kIPaddress = null;
                kMacAddress = null;
        }

        public Outlet (int id, String type, String name, int status, int dimmer, String ip, String
mac) {
                kId = id;
                kType = type;
                kName = name;
                kStatus = status;
                kDimmer = dimmer;
                kIPaddress = ip;
                kMacAddress = mac;
        }

        /**
        * @return the kId
        */
        public int getkId() {
                return kId;
        }

        /**
        * @param kId the kId to set
        */
        public void setkId(int kId) {
                this.kId = kId;
        }

        /**
        * @return the kType
        */
        public String getkType() {
                return kType;
        }

        /**
        * @param kType the kType to set
        */
        public void setkType(String kType) {
                this.kType = kType;
        }

        /**
        * @return the kName
        */
        public String getkName() {
                return kName;
        }

        /**
        * @param kName the kName to set
        */
        public void setkName(String kName) {
                this.kName = kName;
        }

        /**
        * @return the kStatus
        */
        public int getkStatus() {
                return kStatus;
        }

        /**
        * @param kStatus the kStatus to set
        */
        public void setkStatus(int kStatus) {
                this.kStatus = kStatus;
        }
```

```
	/**
	* @return the kDimmer
	*/
	public int getkDimmer() {
		return kDimmer;
	}


	/**
	* @param kDimmer the kDimmer to set
	*/
	public void setkDimmer(int kDimmer) {
		this.kDimmer = kDimmer;
	}


	/**
	* @return the kIPaddress
	*/
	public String getkIPaddress() {
		return kIPaddress;
	}


	/**
	* @param kIPaddress the kIPaddress to set
	*/
	public void setkIPaddress(String kIPaddress) {
		this.kIPaddress = kIPaddress;
	}


	/**
	* @return the kMacAddress
	*/
	public String getkMacAddress() {
		return kMacAddress;
	}


	/**
	* @param kMacAddress the kMacAddress to set
	*/
	public void setkMacAddress(String kMacAddress) {
		this.kMacAddress = kMacAddress;
	}
}



/*
* Node.java - Node object which contains an outlet and the sockets to communicate with
*/
package sp.server;


import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.net.Socket;



public class Node {


	private Socket kSocket;
	private Outlet kOutlet;
	private BufferedInputStream kInStream;
	private BufferedOutputStream kOutStream;


	private boolean stateChanged = false;


	public Node (Socket socket, BufferedInputStream in, BufferedOutputStream out, Outlet outlet) {
		this.kSocket = socket;
		this.kOutlet = outlet;
		kInStream = in;
		kOutStream = out;
	}
```

```
public String toString() {
        return new String("This is node: " + kOutlet.getkId() +
                          " Status is: " + kOutlet.getkStatus() +
                          " With dim vale of: " + kOutlet.getkDimmer());
}


/**
* @return the kSocket
*/
public Socket getkSocket() {
        return kSocket;
}


/**
* @param kSocket the kSocket to set
*/
public void setkSocket(Socket kSocket) {
        this.kSocket = kSocket;
}


/**
* @return the kOutlet
*/
public Outlet getkOutlet() {
        return kOutlet;
}


/**
* @param kOutlet the kOutlet to set
*/
public void setkOutlet(Outlet kOutlet) {
        this.kOutlet = kOutlet;
}


/**
* @return the kInStream
*/
public BufferedInputStream getkInStream() {
        return kInStream;
}


/**
* @param kInStream the kInStream to set
*/
public void setkInStream(BufferedInputStream kInStream) {
        this.kInStream = kInStream;
}


/**
* @return the kOutStream
*/
public BufferedOutputStream getkOutStream() {
        return kOutStream;
}


/**
* @param kOutStream the kOutStream to set
*/
public void setkOutStream(BufferedOutputStream kOutStream) {
        this.kOutStream = kOutStream;
}


/**
* @return the stateChanged
*/
public boolean isStateChanged() {
        return stateChanged;
}


/**
```

```
	* @param stateChanged the stateChanged to set
	*/
	public void setStateChanged(boolean stateChanged) {
		this.stateChanged = stateChanged;
	}


}


/*
* Packet.java - Serializable Java object to send between server and java
*/


package sp.server;


import java.io.Serializable;
import java.util.ArrayList;


import com.sun.tools.javac.util.List;


public class Packet implements Serializable {


	public enum Action {
		SERVER_ACK, TX_TIMEOUT, END_SERVER, WAIT_CLIENT_ACK,
		WAIT_RX_TX, PERFORM_QUERY, DO_NOTHING,
		TURN_ON_FROM_APP, TURN_OFF_FROM_APP;
	}


	private Action mState;
	private int mId;
	private int mEnable;
	private int mDimVal;
	private ArrayList<Outlet> mList;


	public Packet () {
		mState = Action.DO_NOTHING;
		mEnable = Outlet.OFF;
		mDimVal = Outlet.MIN_DIMMER_VAL;
		mId = 0;
		mList = new ArrayList<Outlet>();
	}


	/**
	* @return the mState
	*/
	public Action getmState() {
		return mState;
	}


	/**
	* @param mState the mState to set
	*/
	public void setmState(Action mState) {
		this.mState = mState;
	}


	/**
	* @return the mEnable
	*/
	public int getmEnable() {
		return mEnable;
	}


	/**
	* @param mEnable the mEnable to set
	*/
	public void setmEnable(int mEnable) {
		this.mEnable = mEnable;
	}
```

```
	/**
	* @return the mDimVal
	*/
	public int getmDimVal() {
		return mDimVal;
	}


	/**
	* @param mDimVal the mDimVal to set
	*/
	public void setmDimVal(int mDimVal) {
		this.mDimVal = mDimVal;
	}


	/**
	* @return the mList
	*/
	public ArrayList<Outlet> getmList() {
		return mList;
	}


	/**
	* @param mList the mList to set
	*/
	public void setmList(ArrayList<Outlet> mList) {
		this.mList = mList;
	}


	/**
	* @return the mId
	*/
	public int getmId() {
		return mId;
	}


	/**
	* @param mId the mId to set
	*/
	public void setmId(int mId) {
		this.mId = mId;
	}

}


/*
* Message.java - used to represent the structure to send between server and outlet node
*/
package sp.server;


import java.io.Serializable;
import java.nio.charset.Charset;


public class Message implements Serializable {


	private final static String ASCII = "US-ASCII";


	private final static int DEF_BUFFER_SIZE = 16;


	//Actions
	public final static char ACTION_DEFAULT =  '%';
	public final static char ACTION_TYPE = 'T';


	//Some valid responses
	public final static String NODE_TYPE = "NODE";
	public final static String MOBILE_TYPE = "MOBILE";
```

```
	//instance fields
	public char identifier;
	public char action;
	public byte dataSize;
	public byte[] data;


	public Message() {
		this.identifier = '$';
		this.action = ACTION_DEFAULT;
		this.data = new byte[DEF_BUFFER_SIZE];
		this.dataSize = 0;
	}


	public Message(char action, byte[] data, byte size) {
		this.identifier = '$';
		this.action = action;
		this.data = new byte[size];
		this.data = data;
		this.dataSize = size;
	}


	/*
	* ToString method for the whole Message object, used for debuging and printing.
	*/
	public String toString() {
		String result = "action: " + this.action + " size: " + this.dataSize +  " data: " +
new String (data, 0, dataSize, Charset.forName(ASCII));
		return result;
	}


	/*
	* Returns the valid data in the data buffer as a string, or null.
	*/
	public String dataBufferAsString() {
		if (dataSize <= 0 || null == data) {
			return null;
		}
		else {
			return new String(data, 0, dataSize, Charset.forName(ASCII));
		}
	}


	/*
	* Returns only the valid data, aka dataSize, of the data buffer.
	*/
	public byte[] getValidData() {
		byte[] result = new byte[dataSize];
		result = this.data.clone();
		return result;
	}


	/*
	* Convert message into a byte array to send over the socket.
	*/
	public byte[] toByteDataArray() {
		byte[] result = new byte[DEF_BUFFER_SIZE + 3];
		result[0] = (byte) this.identifier;
		result[1] = (byte) this.action;
		result[2] = (byte) this.dataSize;
		for (int i = 0; i < dataSize; i++) {
			result[3+i] = this.data[i];
		}
		return result;
	}


	/*
	* Convert message into a byte array to send over the socket.
	*/
	public byte[] toByteDataArrayHack() {
		byte[] result = new byte[DEF_BUFFER_SIZE + 12];
		result[0] = '\r';
		result[1] = '\n';
		result[2] = 27;
		result[3] = 'S';
```

```
		result[4] = '0';
		result[5] = (byte) this.identifier;
		result[6] = (byte) this.action;
		result[7] = (byte) this.dataSize;
		int i = 0;
		for (i = 0; i < dataSize; i++) {
			result[8+i] = this.data[i];
		}
		result[8+i] = 27;
		result[8+i+1] = 'E';
		result[8+i+2] = '\r';
		result[8+i+3] = '\n';
		return result;
	}


	/*
	* Static method to convert a byte array to a Message object.
	*/
	public static Message byteToMessage(byte[] buffer, int len) {


		String str = new String(buffer, Charset.forName(ASCII));
		Message result = new Message();
		char[] arr = new char[DEF_BUFFER_SIZE];
		str.getChars(0, len, arr, 0);


		char c = (char)((int)buffer[2]);
		int size = Character.digit(c, 10);


		result.action = arr[1];
		result.dataSize = (byte) size;


		/*
		System.out.println("char: " + c);
		System.out.println("action: " + result.action);
		System.out.println("numread: " + len);
		System.out.println("datasize: " + (int)result.dataSize);
		*/


		for(int i = 0; i < result.dataSize; i++) {
			result.data[i] = buffer[i+3];
		}
		return result;
	}


}
```

ANDROID CODE

```
////////////////////////////////////////////////////////////////////////
//////////                Android Code                        ////////////
////////////////////////////////////////////////////////////////////////
```

```
/*
* MainActivity.java - activity which is used for the main functionality of the app
*/
package sp.remote.app;


import java.util.ArrayList;


import sp.remote.R;
import sp.remote.database.DatabaseHelper;
import sp.remote.database.DatabaseInitializer;
import sp.remote.model.K;
import sp.server.Outlet;
import android.app.ActionBar;
import android.app.ActionBar.OnNavigationListener;
import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentTransaction;
import android.app.ListFragment;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.Configuration;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Rect;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.TouchDelegate;
import android.view.View;
import android.view.View.OnLongClickListener;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.BaseAdapter;
import android.widget.CheckedTextView;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ListView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.widget.SpinnerAdapter;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;


/**
* Demonstration of using fragments to implement different activity layouts.
* This sample provides a different layout (and activity flow) when run in
* landscape.
*/
public class MainActivity extends Activity {


        private SQLiteDatabase mDB;


   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       setContentView(R.layout.fragment_layout);

       //ActionBar Setup
       ActionBar mainActivityAB = getActionBar();
       mainActivityAB.setIcon(R.drawable.icon_ab);
       mainActivityAB.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
```

```
        mainActivityAB.setDisplayOptions(0, ActionBar.DISPLAY_SHOW_TITLE);
        String[] arr = getResources().getStringArray(R.array.navigation_array);
        mainActivityAB.setListNavigationCallbacks(
                  new ActionBarAdapter(this, R.layout.spinner_title_item,
R.layout.spinner_dropdown_item, arr),
                  new NavigationListener());

//         //Start BackgroundService
//         Intent service = new Intent(this.getApplicationContext(), BackgroundService.class);
//         this.getApplicationContext().startService(service);
    }
    @Override
    protected void onResume() {
          super.onResume();
    }

    /*Called to inflate menu from xml resuource*/
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
          MenuInflater inflater = getMenuInflater();
          inflater.inflate(R.menu.main_activity, menu);
          return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
          switch (item.getItemId()) {

          case android.R.id.home:
                  break;

          case R.id.ab_menu_preferences:
                  Intent intent = new Intent(getApplicationContext(), PreferencesActivity.class);
            startActivity(intent);
                  break;
          case R.id.ab_menu_add:
                  Intent temp = new Intent(getApplicationContext(), AddingToListActivity.class);
                  startActivity(temp);
                  break;
          case R.id.ab_menu_discard:
                  Intent serv = new Intent(getApplicationContext(), BackgroundService.class);
                  serv.putExtra(K.SERVICE_ACTION_KEY, K.ACTION_DELETE_DB);
                  startService(serv);
                  break;

          case R.id.ab_menu_refresh:
                  Intent service = new Intent(getApplicationContext(), BackgroundService.class);
                  service.putExtra(K.SERVICE_ACTION_KEY, K.ACTION_REFRESH_DB);
                  startService(service);
                  break;
          default:
                  super.onOptionsItemSelected(item);
          }

          return true;
    }

    public static class ActionBarAdapter extends BaseAdapter implements SpinnerAdapter {

          private final Activity mActivity;
          private String[] mList;
          private final int mListViewId;
          private final int mDropdownViewId;

          public ActionBarAdapter(Activity activity, int listViewResourceId, int dropdownViewResourceId,
String[] list) {
                  mActivity = activity;
                  mListViewId = listViewResourceId;
                  mDropdownViewId = dropdownViewResourceId;
                  mList = list;
          }

          static class ViewHolder {
                  protected CheckedTextView topText;
                  protected CheckedTextView bottomText;
                  protected CheckedTextView topDropdownText;
                  protected CheckedTextView bottomDropdownText;
          }


          @Override
```

```
        public View getView(int position, View convertView, ViewGroup parent) {
                View view = null;
                if (convertView == null) {
                        LayoutInflater inflater = mActivity.getLayoutInflater();
                        view = inflater.inflate(mListViewId, null);
                        ViewHolder viewHolder = new ViewHolder();
                        viewHolder.topText = (CheckedTextView)
view.findViewById(R.id.title_top_text);
                        viewHolder.bottomText = (CheckedTextView)
view.findViewById(R.id.title_bottom_text);
                        view.setTag(viewHolder);
                } else {
                        view = convertView;
                }
                ViewHolder holder = (ViewHolder)view.getTag();
                holder.topText.setText(mList[position]);
                holder.bottomText.setText("Bottom");
                return view;
        }



        @Override
        public View getDropDownView(int position, View convertView, ViewGroup parent) {
                View view = null;
                if (convertView == null) {
                        LayoutInflater inflater = mActivity.getLayoutInflater();
                        view = inflater.inflate(mDropdownViewId, null);
                        ViewHolder viewHolder = new ViewHolder();
                        viewHolder.topDropdownText = (CheckedTextView)
view.findViewById(R.id.spinner_top_text);
                        viewHolder.bottomDropdownText = (CheckedTextView)
view.findViewById(R.id.spinner_bottom_text);
                        view.setTag(viewHolder);
                } else {
                        view = convertView;
                }
                ViewHolder holder = (ViewHolder)view.getTag();
                holder.topDropdownText.setText(mList[position]);
                holder.bottomDropdownText.setText("Bottom");
                return view;
        }



        @Override
        public int getCount() {
                return mList.length;
        }



        @Override
        public Object getItem(int position) {
                return mList[position];
        }



        @Override
        public long getItemId(int position) {
                return position;
        }

   }

   /*
    *NavigationListener for ActionBar to navigate between fragments
    */
   private class NavigationListener implements OnNavigationListener {


                @Override
                public boolean onNavigationItemSelected(int itemPosition, long itemId) {
                        MyLog.d("justin", "ID: " + itemId + " Position: " + itemPosition);
                        return false;
                }
   }
```

```
    /**
     * This is a secondary activity, to show what the user has selected
     * when the screen is not large enough to show it all in one activity.
     */

    public static class DetailsActivity extends Activity {



        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);


            if (getResources().getConfiguration().orientation
                    == Configuration.ORIENTATION_LANDSCAPE) {
                // If the screen is now in landscape mode, we can show the
                // dialog in-line with the list so we don't need this activity.
                finish();
                return;
            }


            if (savedInstanceState == null) {
                // During initial setup, plug in the details fragment.
                DetailsFragment details = new DetailsFragment();
                details.setArguments(getIntent().getExtras());
                getFragmentManager().beginTransaction().add(android.R.id.content, details).commit();
            }

            getActionBar().setDisplayHomeAsUpEnabled(true);
        }
    }



    /**
     * This is the "top-level" fragment, showing a list of items that the
     * user can pick.  Upon picking an item, it takes care of displaying the
     * data to the user as appropriate based on the currrent UI layout.
     */

    public static class TitlesFragment extends ListFragment {
        boolean mDualPane;
        int mCurCheckPosition = 0;
        private ArrayList<Outlet> mPowerList;
        private static SocketAdapter mAdapter;

        @Override
        public void onActivityCreated(Bundle savedInstanceState) {
            super.onActivityCreated(savedInstanceState);
            mPowerList = new ArrayList<Outlet>();



            mAdapter = new SocketAdapter(getActivity(), R.layout.power_item_layout, mPowerList);
            setListAdapter(mAdapter);
            getListView().setItemsCanFocus(true);
            generateDatabaseList(mAdapter);
            // Populate list with our static array of titles.
//              setListAdapter(new ArrayAdapter<String>(getActivity(),
//                      android.R.layout.simple_list_item_1, Shakespeare.TITLES));
//              setListAdapter(new ArrayAdapter<String>(getActivity(),
//                          R.layout.my_simple, Shakespeare.TITLES));


            // Check to see if we have a frame in which to embed the details
            // fragment directly in the containing UI.
            View detailsFrame = getActivity().findViewById(R.id.details);
            mDualPane = detailsFrame != null && detailsFrame.getVisibility() == View.VISIBLE;


            if (savedInstanceState != null) {
                // Restore last state for checked position.
                mCurCheckPosition = savedInstanceState.getInt("curChoice", 0);
```

```
            }


            if (mDualPane) {
                // In dual-pane mode, the list view highlights the selected item.
                getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
                // Make sure our UI is in the correct state.
                showDetails(mCurCheckPosition);
            }
        }

        @Override
        public void onResume() {
          super.onResume();
          MyLog.d("justin", "OnResume()");
          generateDatabaseList((SocketAdapter)getListAdapter());
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {

          View viewer = inflater.inflate(R.layout.list_layout, container, false);
                          return viewer;
                  }



        @Override
        public void onSaveInstanceState(Bundle outState) {
            super.onSaveInstanceState(outState);
            outState.putInt("curChoice", mCurCheckPosition);
        }


        @Override
        public void onListItemClick(ListView l, View v, int position, long id) {
            showDetails(position);
        }

        public static SocketAdapter getSocketAdapter() {
          return mAdapter;
        }




        /**
         * Helper function to show the details of a selected item, either by
         * displaying a fragment in-place in the current UI, or starting a
         * whole new activity in which it is displayed.
         */
        void showDetails(int index) {
            mCurCheckPosition = index;


            if (mDualPane) {
                // We can display everything in-place with fragments, so update
                // the list to highlight the selected item and show the data.
                getListView().setItemChecked(index, true);


                // Check what fragment is currently shown, replace if needed.
                DetailsFragment details = (DetailsFragment)
                        getFragmentManager().findFragmentById(R.id.details);
                if (details == null || details.getShownIndex() != index) {
                    // Make new fragment to show this selection.
                    details = DetailsFragment.newInstance(index);


                    // Execute a transaction, replacing any existing fragment
                    // with this one inside the frame.
                    FragmentTransaction ft = getFragmentManager().beginTransaction();
                    ft.replace(R.id.details, details);
                    ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
                    ft.commit();
                }


            } else {
                // Otherwise we need to launch a new activity to display
```

```
                // the dialog fragment with selected text.
                Intent intent = new Intent();
                intent.setClass(getActivity(), DetailsActivity.class);
                intent.putExtra("index", index);
                startActivity(intent);
            }
        }

        void generateDatabaseList(SocketAdapter adapter) {
          MyLog.d("justin", "in database list");
            DatabaseInitializer dbi = new
DatabaseInitializer(this.getActivity().getApplicationContext());
            SQLiteDatabase db = dbi.getReadableDatabase();
            Cursor cur = DatabaseHelper.getAllEntries(db, dbi.getDatabaseName());
            MyLog.d("justin", "count: " + cur.getCount());
            adapter.clear();
            while(cur.moveToNext()) {
                  //list.add(new Power(status, macAddress, ipAddress, name));


                  adapter.add(new Outlet(
                                  cur.getInt(cur.getColumnIndex("_id")),
                                  cur.getString(cur.getColumnIndex("_type")),
                                  cur.getString(cur.getColumnIndex("_name")),
                                  cur.getInt(cur.getColumnIndex("_status")),
                                  cur.getInt(cur.getColumnIndex("_dimVal")),
                                  cur.getString(cur.getColumnIndex("_ip")),
                                  cur.getString(cur.getColumnIndex("_mac"))));


            }
            db.close();
            cur = null;
            db = null;
        }

        public void myOnClickMethod(View v) {
          MyLog.d("tag", "pressed");
        }

        /*
         * Private class for the adapter to listView
         */
        private class SocketAdapter extends ArrayAdapter<Outlet> {

          private final Activity mContext;
          private ArrayList<Outlet> mSocketList;


          public SocketAdapter(Activity context, int textViewResourceId, ArrayList<Outlet> list) {
                  super(context, textViewResourceId, list);
                  mContext = context;
                  mSocketList = list;
          }

          class ViewHolder {
                  protected TextView nameText;
                  protected TextView roomText;
                  protected Switch toggleSwitch;
                  protected SeekBar seekBar;
          }



          @Override
          public View getView(int pos, View convertView, ViewGroup parent) {
                  View view = null;
                  if (convertView == null) {
                          LayoutInflater inflator = mContext.getLayoutInflater();
                          view = inflator.inflate(R.layout.power_item_layout, null);
                          final ViewHolder viewHolder = new ViewHolder();
                          viewHolder.nameText = (TextView) view.findViewById(R.id.itemLabel);
                          viewHolder.roomText = (TextView) view.findViewById(R.id.statusLabel);
                          viewHolder.toggleSwitch = (Switch) view.findViewById(R.id.itemSwitch);
                          viewHolder.seekBar = (SeekBar) view.findViewById(R.id.seekBarList);
                          view.setTag(viewHolder);

                  } else {
                          view = convertView;
                  }
                  final ViewHolder holder = (ViewHolder)view.getTag();
```

```
                holder.nameText.setText(mSocketList.get(pos).getkName());
                holder.roomText.setText("Living room");
                //get parent view to set touchDelegate
                final View root = view.findViewById(R.id.power_item_layout);
                //TouchDelegate for checkbox;
                root.post(new Runnable() {


                        @Override
                        public void run() {
                                final Rect r = new Rect();
                                holder.toggleSwitch.getHitRect(r);
                                r.top -= 20;
                                r.bottom += 20;
                                r.right += 30;
                                r.left -= 30;
                                root.setTouchDelegate(new TouchDelegate(r, holder.toggleSwitch));
                        }
                });


                holder.seekBar.setOnSeekBarChangeListener(new ProgressSeekBarListener(pos,
holder.toggleSwitch));

                holder.toggleSwitch.setOnCheckedChangeListener(new CheckChangeListener(pos,
holder.seekBar));
                holder.toggleSwitch.setOnLongClickListener(new OnLongClickListener() {


                                        @Override
                                        public boolean onLongClick(View v) {
                                                Toast.makeText(mContext, "Turn outlet on/off",
Toast.LENGTH_SHORT).show();
                                                return true;
                                        }
                                });

                //Updating UI
                holder.toggleSwitch.setChecked((mSocketList.get(pos).getkStatus() == K.STATUS_ON) ?
true : false);

                //Allowing ListView item to actually be clicked
                /*
                root.setNextFocusRightId(holder.toggleSwitch.getId());
                root.setClickable(true);
                root.setFocusable(true);
                root.setBackgroundResource(android.R.drawable.menuitem_background);
                */


                return view;
        }

        private class ProgressSeekBarListener implements OnSeekBarChangeListener {

                private int pos;
                private Switch button;

                public ProgressSeekBarListener (int posititon, Switch switchBbutton) {
                        pos = posititon;
                        button = switchBbutton;
                }


                                @Override
                                public void onProgressChanged(SeekBar seekBar, int progress,
                                                boolean fromUser) {
                                        SharedPreferences sharedPrefs =
PreferenceManager.getDefaultSharedPreferences(getActivity().getApplicationContext());
                                        boolean kian_mode =
sharedPrefs.getBoolean("pg_checkbox_key", false);
                                        if (kian_mode) {
                                        int value = K.ACTION_TURN_ON;
                                        int dimVal = 248 - seekBar.getProgress();
                                        MyLog.d("justin", "Seekbar val: " + progress + " -> DimVal:
" + dimVal);
                                Intent service = new Intent(getContext(), BackgroundService.class);
                                service.putExtra(K.SERVICE_ACTION_KEY, value);
                                service.putExtra(K.SERVICE_ID_KEY, pos);
                                service.putExtra(K.SERVICE_DIM_VAL_KEY, dimVal);
                                getContext().startService(service);
```

```
                                button.setChecked(true);
                                        }


                                 }


                                 @Override
                                 public void onStartTrackingTouch(SeekBar seekBar) {
                                         // TODO Auto-generated method stub


                                 }


                                 @Override
                                 public void onStopTrackingTouch(SeekBar seekBar) {


                                         int value = K.ACTION_TURN_ON;
                                         int dimVal = 248 - seekBar.getProgress();
                                         MyLog.d("justin", "Seekbar val: " + seekBar.getProgress() +
" -> DimVal: " + dimVal);
                                Intent service = new Intent(getContext(), BackgroundService.class);
                                service.putExtra(K.SERVICE_ACTION_KEY, value);
                                service.putExtra(K.SERVICE_ID_KEY, pos);
                                service.putExtra(K.SERVICE_DIM_VAL_KEY, dimVal);
                                getContext().startService(service);
                                button.setChecked(true);

                                 }

        }

        private class CheckChangeListener implements OnCheckedChangeListener {

                private int index;
                private SeekBar seekBar;
                public CheckChangeListener(int pos, SeekBar bar) {
                        index = pos;
                        seekBar = bar;
                }
                                 @Override
                                 public void onCheckedChanged(CompoundButton buttonView,
                                                 boolean isChecked) {
                                         switch (buttonView.getId()) {


                                                 case R.id.itemSwitch:
                                                         MyLog.d("justin", "toggleSwitch: " +
String.valueOf(index) + " - " + String.valueOf(isChecked));
                                                         DatabaseInitializer dbi = new
DatabaseInitializer(mContext);

                                                         SQLiteDatabase db = dbi.getWritableDatabase();
                                                         mSocketList.get(index).setkStatus(isChecked ?
K.STATUS_ON : K.STATUS_OFF);

                                                         int count = DatabaseHelper.updateLight(db,
dbi.getDatabaseName(), mSocketList.get(index));
                                                         MyLog.d("justin", "num rows affected: " + count);
                                                         db.close();


                                                         int value = K.ACTION_TURN_ON;
                                                 if (!buttonView.isChecked()) value = K.ACTION_TURN_OFF;
                                                 Intent service = new Intent(getContext(),
BackgroundService.class);
                                                 service.putExtra(K.SERVICE_ACTION_KEY, value);
                                                 service.putExtra(K.SERVICE_ID_KEY, index);
                                                 service.putExtra(K.SERVICE_DIM_VAL_KEY, 248 -
seekBar.getProgress());
                                                 getContext().startService(service);


                                                         break;
                                                   default:
                                                         MyLog.d("justin", "No action for ID");
                                                   }
                                 }
        }

       }
```

```
}



/**
 * This is the secondary fragment, displaying the details of a particular
 * item.
 */


public static class DetailsFragment extends Fragment {
    /**
     * Create a new instance of DetailsFragment, initialized to
     * show the text at 'index'.
     */
    public static DetailsFragment newInstance(int index) {
        DetailsFragment f = new DetailsFragment();


        // Supply index input as an argument.
        Bundle args = new Bundle();
        args.putInt("index", index);
        f.setArguments(args);


        return f;
    }


    public int getShownIndex() {
        return getArguments().getInt("index", 0);
    }


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        if (container == null) {
            // We have different layouts, and in one of them this
            // fragment's containing frame doesn't exist.  The fragment
            // may still be created from its saved state, but there is
            // no reason to try to create its view hierarchy because it
            // won't be displayed.  Note this is not needed -- we could
            // just run the code below, where we would create and return
            // the view hierarchy; it would just never be used.
            return null;
        }
        View view = inflater.inflate(R.layout.info_layout, null);
        SeekBar seek = (SeekBar)view.findViewById(R.id.seekBar);
        TextView nameText = (TextView)view.findViewById(R.id.nameTextView);
        TextView statusText = (TextView)view.findViewById(R.id.statusTextView);
        if (seek == null)
              MyLog.d("justin", "seekbar is null");
        seek.setOnSeekBarChangeListener(new SeekBarListener(statusText, nameText));
        return view;
        /*
        ScrollView scroller = new ScrollView(getActivity());
        TextView text = new TextView(getActivity());
        int padding = (int)TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
                4, getActivity().getResources().getDisplayMetrics());
        text.setPadding(padding, padding, padding, padding);
        scroller.addView(text);
        text.setText(Shakespeare.DIALOGUE[getShownIndex()]);
        return scroller;
        */
    }

    private class SeekBarListener implements OnSeekBarChangeListener {


      private TextView statusText;
      private TextView progressText;

      public SeekBarListener(TextView statusText, TextView progressText) {
              this.statusText = statusText;
              this.progressText = progressText;
      }

              @Override
```

```
                public void onProgressChanged(SeekBar seekBar, int progress,
                                boolean fromUser) {
                        progressText.setText(String.valueOf(progress));
                }


                @Override
                public void onStartTrackingTouch(SeekBar seekBar) {
                        statusText.setText("tracking on");
                }


                @Override
                public void onStopTrackingTouch(SeekBar seekBar) {
                        statusText.setText("tracking off");
                }

        }

    }

}
```

```
/*
* BackgroundService.java - Service which runs on a background thread to perform database queries and
socket
* communication
*/


package sp.remote.app;


import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OptionalDataException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.nio.charset.Charset;
import java.util.ArrayList;


import sp.remote.database.DatabaseInitializer;
import sp.remote.model.K;
import sp.server.Outlet;
import sp.server.Packet;
import sp.server.Packet.Action;
import android.app.Activity;
import android.app.Service;
import android.content.ContentValues;
import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;


public class BackgroundService extends Service {


         private final static String tag = "BackgroundService";
         private final static String handshake = "JKE";
         private final static String handshakeResponse = "EKJ";
         private final static String ASCII = "US-ASCII";
         private final static int SERVICE_PORT_NUM = 52937;
         private final static int EINVALID = -321;
         private final static int UDP_TIMEOUT = 2000; //2 secs


         //instance fields
```

```
      InetAddress mServerAddress;
int mServerPort;
Socket mConnection = null;
BufferedOutputStream mOut = null;
      ObjectInputStream mInTCP = null;
      ObjectOutputStream mOutTCP = null;
      public ArrayList<Outlet> mOutletList;


      //used to receive messages from the Activity
      final Messenger mInMessenger = new Messenger(new IncomingHandler());


      //used to send messages to the Activity
      private Messenger mOutMessenger;


      @Override
      public IBinder onBind(Intent intent) {
               Bundle extras = intent.getExtras();
               //get messanger from the Activity
               if (extras != null) {
                        mOutMessenger = (Messenger) extras.get(K.MESSENGER);
               }
               return mInMessenger.getBinder();
      }


      @Override
      public void onCreate() {
               super.onCreate();
               MyLog.d(tag, "Service started");
      }


      @Override
public int onStartCommand(Intent intent, int flags, int startId) {

               MyLog.i(tag, "Received start id " + startId + ": " + intent);
    Bundle extras = intent.getExtras();
    int action = 0;
    final int id = 0;

    if (null != extras) {
      action = extras.getInt(K.SERVICE_ACTION_KEY, 0);

    }
    final int dimVal = extras.getInt(K.SERVICE_DIM_VAL_KEY, 0);

    switch (action) {

    case K.ACTION_CONNECT_SERVER:
      MyLog.d(tag, "Server connected: " + String.valueOf(isConnectionAlive()));
      if (!isConnectionAlive()) {
               new Thread() {
                        @Override
                        public void run() {
                                 boolean connected = connectToServer();
                                 MyLog.d(tag, "Connected: " + String.valueOf(connected));

                        }
               }.start();
      }
      break;

    case K.ACTION_TURN_ON:

      if (isConnectionAlive()) {
               new Thread() {
                        @Override
                        public void run() {

                                 Packet packet = new Packet();
                                 packet.setmState(Action.TURN_ON_FROM_APP);
                                 packet.setmEnable(Outlet.ON);
                                 packet.setmId(id);
                                 packet.setmDimVal(dimVal);
                                 packet.setmList(null);
                                 sendPacket(packet);
                        }
               }.start();
```

```
		}
		break;

	case K.ACTION_TURN_OFF:
		MyLog.d(tag, "TURN OFF");
		MyLog.d(tag, "dim val: " + dimVal);
			new Thread() {
				@Override
				public void run() {
					// if isConnectionAlive is TRUE, connectToServer won't run
					if (isConnectionAlive() || connectToServer()) {
						Packet packet = new Packet();
						packet.setmState(Action.TURN_OFF_FROM_APP);
					packet.setmEnable(Outlet.OFF);
					packet.setmId(id);
					packet.setmDimVal(dimVal);
					packet.setmList(null);
					sendPacket(packet);
					}
				}
			}.start();
		break;

	case K.ACTION_DELETE_DB:
		new Thread() {
				@Override
				public void run() {
					DatabaseInitializer dbi = new
DatabaseInitializer(getApplicationContext());
					SQLiteDatabase db = dbi.getWritableDatabase();
					String delete = "DELETE FROM " + dbi.TABLE_NAME + ";";
					db.execSQL(delete);
				}
			}.start();
		break;

	case K.ACTION_REFRESH_DB:
		if (isConnectionAlive()) {
			new Thread() {
				@Override
				public void run() {
					Packet packet = new Packet();
					packet.setmState(Action.PERFORM_QUERY);
					sendPacket(packet);

					try {
							packet = (Packet)mInTCP.readObject();


							for (Outlet outlet : packet.getmList()) {
								//mOutletList.add(outlet);
								addOutletToDatabase(outlet);
							}
					} catch (OptionalDataException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					} catch (ClassNotFoundException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					} catch (IOException e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}


				}
			}.start();
		}
		break;

	case K.ACTION_DISCONNECT_SERVER:
		closeConnections();
		break;

	case K.ACTION_ACK:
		if (isConnectionAlive()) {
			new Thread() {
				@Override
				public void run() {

					Packet packet = new Packet();
```

```
                                packet.setmState(Action.SERVER_ACK);
                                sendPacket(packet);
                        }
                }.start();
        }
        break;

    default:
        break;
    }
    // We want this service to continue running until it is explicitly
    // stopped, so return sticky.
    return START_STICKY;
}


    @Override
    public void onDestroy() {
            super.onDestroy();
            MyLog.d(tag, "Service destroyed");
            closeConnections();
            mServerAddress = null;
            mServerPort = EINVALID;
    }


    /*
    * Helper to insert an Outlet object in to database
    */
    public void addOutletToDatabase(Outlet outlet) {
            DatabaseInitializer dbi = new DatabaseInitializer(getApplicationContext());
            SQLiteDatabase db = dbi.getWritableDatabase();
            /*
            String insert = "INSERT INTO " + dbi.getDatabaseName() +
                            " (_id, _type, _name, _staus, _dimVal, _ip, _mac) VALUES (" +
                            outlet.getkId() + ", " +
                            "'" + outlet.getkType() + "', " +
                            "'" + outlet.getkName() + "', " +
                            outlet.getkStatus() + ", " +
                            outlet.getkDimmer() + ", " +
                            "'" + outlet.getkIPaddress() + "', " +
                            "'" + outlet.getkMacAddress() + "');";
            */
            ContentValues cv = new ContentValues();
            cv.put("_id", outlet.getkId());
            cv.put("_type", outlet.getkType());
            cv.put("_name", outlet.getkName());
            cv.put("_status", outlet.getkStatus());
            cv.put("_dimVal", outlet.getkDimmer());
            cv.put("_ip", outlet.getkIPaddress());
            cv.put("_mac", "mac");


            if (-1 == db.insert(dbi.getDatabaseName(), null, cv))
                    MyLog.d("justin", "SQL error");


            db.close();
    }


    /*
    * Connect to Server on a separate thread using UDP and then TCP to communicate
    * Returns whether or not the connection was established
    */
    private boolean connectToServer() {


            boolean done = false;
            int maxTCPTries = 2;
            int maxUDPBroadcastTries = 2;
            do {
                    if (null != mServerAddress && mServerPort != EINVALID) {
                            if (!(done = startTCPConnection())) {
                                    performBroadcast();
                            }
                            maxTCPTries--;
                    } else {
                            performBroadcast();
                            maxUDPBroadcastTries--;
                    }
```

```
                } while (!done && maxTCPTries > 0 && maxUDPBroadcastTries > 0);
                return done;
        }


        /*
        * Sets up TCP connection with server, call this after performBroadcast
        */
        private boolean startTCPConnection() {
                MyLog.d(tag, "Start TCP Connection");


                try {
                        mConnection = new Socket(InetAddress.getByName(mServerAddress.getHostName()),
mServerPort);
                        MyLog.d(tag, "Bound?: " + mConnection.isBound() + " Connected?: " +
mConnection.isConnected());
                        MyLog.d(tag, "Host port: " + mConnection.getPort() + " name: " +
mConnection.getInetAddress().getCanonicalHostName());
                        //mConnection.setKeepAlive(true);
                        mOut = new BufferedOutputStream(mConnection.getOutputStream());
                        String str = new String("$T3NOT");
                        MyLog.d(tag, str);
                        mOut.write(str.getBytes(Charset.forName(ASCII)));
                        mOut.flush();
                        MyLog.d(tag, "Flushed stream");
                        mOutTCP = new ObjectOutputStream(mOut);
                        mOutTCP.flush();
                        mInTCP = new ObjectInputStream(mConnection.getInputStream());


                } catch (IOException ex) {
                        MyLog.e(tag, "Could not connect: " + mServerAddress.getHostAddress() + "
port: " + mServerPort);
                        return false;
                }
                MyLog.d(tag, "Done with TCP init");
                return mConnection.isConnected();
        }


//      private void sendToServer(String str, boolean waitForResponse) {
//              mOut.print(str);
//              mOut.flush();
//              if (waitForResponse) {
//                      String response = null;
//                      try {
//                              response = min.readLine();
//                      } catch (IOException e) {
//                              // TODO Auto-generated catch block
//                              e.printStackTrace();
//                      }
//                      Toast.makeText(getApplicationContext(), "Response: " + response,
Toast.LENGTH_LONG).show();
//              }
//      }


        private void sendPacket(Packet packet) {
                try {
                        if (null == mOut) MyLog.d(tag, "FUCK YOU");
                        mOutTCP.writeObject(packet);
                        mOutTCP.flush();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }


        }



        /*
        * Uses UDP datagram to find IP address and Port for TCP socket connection
        * Returns server port number or EINVALID
        */
        private void performBroadcast() {


        byte[] buff = new byte[256];
```

```
        DatagramSocket broadcast = null;
        DatagramPacket datagram  = null;
        try {
                broadcast = new DatagramSocket();
                buff = handshake.getBytes(Charset.forName(ASCII));
                datagram = new DatagramPacket(buff, buff.length);
                        datagram.setAddress(InetAddress.getByAddress(new byte[] { (byte) 255, (byte)
255, (byte) 255, (byte) 255 }));
                        datagram.setPort(SERVICE_PORT_NUM);
                        broadcast.setBroadcast(true); //enable broadcast to all
                        //broadcast.setSoTimeout(UDP_TIMEOUT);
                        broadcast.send(datagram);
                        MyLog.d(tag, "Sent, now waiting");
                        //clear buff
                        buff = new byte[256];
                        datagram = new DatagramPacket(buff, buff.length);
                        broadcast.receive(datagram); //blocking until receive response
                        String str = new String(datagram.getData(), 0, datagram.getLength(),
Charset.forName(ASCII));
                        MyLog.d(tag, "Received response: " + str);
                        mServerAddress = datagram.getAddress();
                        MyLog.d(tag, "Recieved from: " + mServerAddress.getHostAddress());
                        //parse response and get port number
                        mServerPort = validateResponseHandshake(str);


                } catch (UnknownHostException e) {
                        e.printStackTrace();
                        MyLog.e(tag, "Error setting the datagram address");
                } catch (SocketException e) {
                        e.printStackTrace();
                        MyLog.e(tag, "Cannot create new DatagramSocket");
                } catch (IOException e) {
                        e.printStackTrace();
                        MyLog.e(tag, "Error sending datagram via UDP socket");
                } finally {
                        if (null != broadcast)
                                broadcast.close();
                }


   }


        /*
        * Confirms handshake and port number response from UDP server
        */
        private int validateResponseHandshake(String str) {
        if (null == str) {
                MyLog.d(tag, "INVALID response from broadcast service");
                return EINVALID;
        }
        //check for proper response
        if (str.substring(0, str.indexOf('@')).equals(handshake + handshakeResponse)) {
                int port = 0;
                try {
                        port = Integer.parseInt(str.substring(str.indexOf('@') + 1, str.length()));
                } catch (NumberFormatException ex) {
                        MyLog.d(tag, "Not a number in: " + str);
                        //ex.printStackTrace();
                        port = EINVALID;
                }
                return port;

        } else {
                MyLog.d(tag, "INVALID response from broadcast service");
                return EINVALID;
        }
   }


        public int getServerPort() {
                return mServerPort;
        }


        private boolean isConnectionAlive() {
                if (null != mConnection)
                        return (mConnection.isConnected() && !mConnection.isClosed() &&
!mConnection.isInputShutdown() && !mConnection.isOutputShutdown());
                else
```

```
                return false;
        }



        private void closeConnections() {
                if (null != mConnection) {
                        if (null != mOutTCP)
                                try {
                                        mOutTCP.close();
                                } catch (IOException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }
                        if (null != mInTCP) {
                                try {
                                        mInTCP.close();
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                        try {
                                mConnection.close();
                                MyLog.d(tag, "mConnection.close()");
                                //Toast.makeText(getApplicationContext(), "Server connection
closed", Toast.LENGTH_LONG).show();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }




        class IncomingHandler extends Handler {
                @Override
                public void handleMessage(Message msg) {
                        MyLog.d(tag, "InHandler received msg");
                        Bundle data = msg.getData();
                        int actionVal = data.getInt(K.SERVICE_ACTION_KEY);


                        switch (actionVal) {
                        case K.ACTION_REFRESH_DB:


                                //refresh the local database from server
                                Message response = Message.obtain();
                                response.arg1 = Activity.RESULT_OK;
                                Bundle bundle = new Bundle();
                                bundle.putInt(K.ACTIVITY_RESPONSE_KEY, K.RESPONSE_REFRESH_DB);
                                response.setData(bundle);
                                try {
                                        mOutMessenger.send(response);
                                        MyLog.d(tag, "Sent to Activity Messenger");
                                } catch (RemoteException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                                break;
                        default:
                                MyLog.d(tag, "action_value: " + actionVal);


                                break;
                        }


                }
        }
```

```
}



/*
* AddingToListActivity.java - testing activity to add a fake node to the system
*/


package sp.remote.app;


import sp.remote.R;
import sp.remote.model.K;
import sp.remote.database.DatabaseInitializer;
import sp.server.Outlet;
import android.app.Activity;
import android.content.ContentValues;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;


public class AddingToListActivity extends Activity {


         private static final String tag = "JC";
         private Button done;
         private EditText nameText;
         private EditText statusText;
         private EditText typeText;


         @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       MyLog.d(tag, "++ OnCreate ++");

       setContentView(R.layout.temp_add);

       nameText = (EditText)findViewById(R.id.editTextName);
       statusText = (EditText)findViewById(R.id.editTextStatus);
       typeText = (EditText)findViewById(R.id.editTextName);
       done = (Button)findViewById(R.id.buttonDone);

       done.setOnClickListener(new MyButtonListener(this));




   }
         @Override
         public void onDestroy() {
       super.onDestroy();
       //android.os.Process.killProcess(android.os.Process.myPid());
   }


         private class MyButtonListener implements OnClickListener {


                  private Activity mContext;
                  public MyButtonListener(Activity thisActivity) {
                           mContext = thisActivity;
                  }


                  @Override
                  public void onClick(View v) {
                           switch (v.getId()) {
                           case R.id.buttonDone:


                                    //db.execSQL("INSERT INTO " + TABLE_NAME + " (_type, _status) VALUES
```

```
(111, 1);");
                                        int type = 0;
                                        int status = 0;
                                        if (typeText.getText().toString().toUpperCase().equals("SOCKET")) {
                                                type = K.POWER_TYPE_SOCKET;
                                        } else if
(typeText.getText().toString().toUpperCase().equals("SWITCH")) {
                                                type = K.POWER_TYPE_SWITCH;
                                        } else {
                                                type = -889;
                                        }


                                        if (Integer.parseInt(statusText.getText().toString()) ==
K.STATUS_OFF) {
                                                status = K.STATUS_OFF;
                                        } else if (Integer.parseInt(statusText.getText().toString()) ==
K.STATUS_ON) {
                                                status = K.STATUS_ON;
                                        } else {
                                                status = Integer.parseInt(statusText.getText().toString());
                                        }


                                DatabaseInitializer dbi = new DatabaseInitializer(getApplicationContext());
                                SQLiteDatabase db = dbi.getWritableDatabase();
                                String insert = "INSERT INTO " + dbi.getDatabaseName() + " (_type, _status,
_name, _ip) VALUES (" + type + ", " + status + ", '" + nameText.getText().toString() + "',
'192.168.0.1');";
                                Outlet outlet = new Outlet(1, "Dimmer", "Name", 1, 120, "192.168.0.1", "MAC");
                                ContentValues cv = new ContentValues();
                                        cv.put("_id", outlet.getkId());
                                        cv.put("_type", outlet.getkType());
                                        cv.put("_name", outlet.getkName());
                                        cv.put("_status", outlet.getkStatus());
                                        cv.put("_dimVal", outlet.getkDimmer());
                                        cv.put("_ip", outlet.getkIPaddress());
                                        cv.put("_mac", outlet.getkMacAddress());


                                        if (-1 == db.insert(dbi.getDatabaseName(), null, cv))
                                                MyLog.d("justin", "SQL error");

                                db.close();
                                MyLog.d("justin", insert);
                                Toast.makeText(mContext, insert, Toast.LENGTH_LONG).show();
                                        break;
                                  }
                        }
                }


/*
* PreferenceActivity.java - Activity for application settings
*/


package sp.remote.app;


import android.app.ActionBar;
import android.app.ActivityManager;
import android.app.AlertDialog;
import android.app.admin.DeviceAdminReceiver;
import android.app.admin.DevicePolicyManager;
import android.content.ComponentName;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.preference.CheckBoxPreference;
import android.preference.EditTextPreference;
import android.preference.ListPreference;
import android.preference.Preference;
import android.preference.Preference.OnPreferenceChangeListener;
import android.preference.Preference.OnPreferenceClickListener;
import android.preference.PreferenceActivity;
import android.preference.PreferenceCategory;
import android.preference.PreferenceFragment;
import android.preference.PreferenceScreen;
import android.text.TextUtils;
```

```
import android.util.Log;
import android.widget.Toast;


import sp.remote.R;
import sp.remote.tcpip.ClientActivity;


import java.util.List;


/**
* This activity provides a comprehensive UI for exploring and operating the DevicePolicyManager
* api.  It consists of two primary modules:
*
* 1:  A device policy controller, implemented here as a series of preference fragments.  Each
*     one contains code to monitor and control a particular subset of device policies.
*
* 2:  A DeviceAdminReceiver, to receive updates from the DevicePolicyManager when certain aspects
*     of the device security status have changed.
*/
public class PreferencesActivity extends PreferenceActivity {


   // Miscellaneous utilities and definitions
   private static final String TAG = "DeviceAdminSample";


   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);



       ActionBar ab = getActionBar();
       ab.setTitle(R.string.ab_title_preferences);
   }


   /**
    * We override this method to provide PreferenceActivity with the top-level preference headers.
    */
   @Override
   public void onBuildHeaders(List<Header> target) {
       loadHeadersFromResource(R.xml.preference_headers, target);
   }


   /**
    * Common fragment code for DevicePolicyManager access.  Provides two shared elements:
    *
    *   1.  Provides instance variables to access activity/context, DevicePolicyManager, etc.
    *   2.  Provides support for the "set password" button(s) shared by multiple fragments.
    */
   public static class AdminSampleFragment extends PreferenceFragment
           implements OnPreferenceChangeListener, OnPreferenceClickListener{


       // Useful instance variables
       protected PreferencesActivity mActivity;
       protected DevicePolicyManager mDPM;
       protected ComponentName mDeviceAdminSample;
       protected boolean mAdminActive;


       // Optional shared UI
       private PreferenceScreen mSetPassword;
       private EditTextPreference mResetPassword;


       @Override
       public void onActivityCreated(Bundle savedInstanceState) {
           super.onActivityCreated(savedInstanceState);
       }




       @Override
       public void onResume() {
           super.onResume();
```

```
        }


        @Override
        public boolean onPreferenceClick(Preference preference) {

            return false;
        }


        @Override
        public boolean onPreferenceChange(Preference preference, Object newValue) {

            return false;
        }


    }


    /**
     * PreferenceFragment for "general" preferences.
     */
    public static class GeneralFragment extends AdminSampleFragment
            implements OnPreferenceChangeListener {
        // UI elements
        private CheckBoxPreference mEnableCheckbox;
        private CheckBoxPreference mDisableCameraCheckbox;


        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.preferences);
//            mEnableCheckbox = (CheckBoxPreference) findPreference(KEY_ENABLE_ADMIN);
//            mEnableCheckbox.setOnPreferenceChangeListener(this);
//            mDisableCameraCheckbox = (CheckBoxPreference) findPreference(KEY_DISABLE_CAMERA);
//            mDisableCameraCheckbox.setOnPreferenceChangeListener(this);
        }


        // At onResume time, reload UI with current values as required
        @Override
        public void onResume() {
            super.onResume();

        }


        @Override
        public boolean onPreferenceChange(Preference preference, Object newValue) {
            if (super.onPreferenceChange(preference, newValue)) {
                return true;
            }
//            boolean value = (Boolean) newValue;
//            if (preference == mEnableCheckbox) {
//                if (value != mAdminActive) {
//                    if (value) {
//                        // Launch the activity to have the user enable our admin.
//                        Intent intent = new Intent(DevicePolicyManager.ACTION_ADD_DEVICE_ADMIN);
//                        intent.putExtra(DevicePolicyManager.EXTRA_DEVICE_ADMIN, mDeviceAdminSample);
//                        intent.putExtra(DevicePolicyManager.EXTRA_ADD_EXPLANATION,
//                                mActivity.getString(R.string.add_admin_extra_app_text));
//                        startActivityForResult(intent, REQUEST_CODE_ENABLE_ADMIN);
//                        // return false - don't update checkbox until we're really active
//                        return false;
//                    } else {
//                        mDPM.removeActiveAdmin(mDeviceAdminSample);
//                        enableDeviceCapabilitiesArea(false);
//                        mAdminActive = false;
//                    }
//                }
//            } else if (preference == mDisableCameraCheckbox) {
//                mDPM.setCameraDisabled(mDeviceAdminSample, value);
//                reloadSummaries();
//            }
            return true;
        }
```

```
    }


    /**
     * PreferenceFragment for "password quality" preferences.
     */
    public static class NetworkFragment extends AdminSampleFragment
            implements OnPreferenceChangeListener {


        // Password quality values
        // This list must match the list found in samples/ApiDemos/res/values/arrays.xml
        final static int[] mPasswordQualityValues = new int[] {
            DevicePolicyManager.PASSWORD_QUALITY_UNSPECIFIED,
            DevicePolicyManager.PASSWORD_QUALITY_SOMETHING,
            DevicePolicyManager.PASSWORD_QUALITY_NUMERIC,
            DevicePolicyManager.PASSWORD_QUALITY_ALPHABETIC,
            DevicePolicyManager.PASSWORD_QUALITY_ALPHANUMERIC,
            DevicePolicyManager.PASSWORD_QUALITY_COMPLEX
        };


        // Password quality values (as strings, for the ListPreference entryValues)
        // This list must match the list found in samples/ApiDemos/res/values/arrays.xml
        final static String[] mPasswordQualityValueStrings = new String[] {
            String.valueOf(DevicePolicyManager.PASSWORD_QUALITY_UNSPECIFIED),
            String.valueOf(DevicePolicyManager.PASSWORD_QUALITY_SOMETHING),
            String.valueOf(DevicePolicyManager.PASSWORD_QUALITY_NUMERIC),
            String.valueOf(DevicePolicyManager.PASSWORD_QUALITY_ALPHABETIC),
            String.valueOf(DevicePolicyManager.PASSWORD_QUALITY_ALPHANUMERIC),
            String.valueOf(DevicePolicyManager.PASSWORD_QUALITY_COMPLEX)
        };


        // UI elements
        private PreferenceCategory mQualityCategory;
        private ListPreference mPasswordQuality;
        private EditTextPreference mMinLength;
        private EditTextPreference mMinLetters;
        private EditTextPreference mMinNumeric;
        private EditTextPreference mMinLowerCase;
        private EditTextPreference mMinUpperCase;
        private EditTextPreference mMinSymbols;
        private EditTextPreference mMinNonLetter;


        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.preferences);

                    Intent client = new Intent(getActivity().getApplicationContext(),
ClientActivity.class);
                    startActivity(client);

                    getActivity().finish();



        }


        @Override
        public void onResume() {
            super.onResume();

        }


        /**
         * Update the summaries of each item to show the local setting and the global setting.
         */
//        @Override
//        protected void reloadSummaries() {
//            super.reloadSummaries();
//            // Show numeric settings for each policy API
//            int local, global;
////            local = mDPM.getPasswordQuality(mDeviceAdminSample);
////            global = mDPM.getPasswordQuality(null);
////            mPasswordQuality.setSummary(
////                    localGlobalSummary(qualityValueToString(local),
```

```
qualityValueToString(global)));
////            local = mDPM.getPasswordMinimumLength(mDeviceAdminSample);
////            global = mDPM.getPasswordMinimumLength(null);
////            mMinLength.setSummary(localGlobalSummary(local, global));
////            local = mDPM.getPasswordMinimumLetters(mDeviceAdminSample);
////            global = mDPM.getPasswordMinimumLetters(null);
////            mMinLetters.setSummary(localGlobalSummary(local, global));
////            local = mDPM.getPasswordMinimumNumeric(mDeviceAdminSample);
////            global = mDPM.getPasswordMinimumNumeric(null);
////            mMinNumeric.setSummary(localGlobalSummary(local, global));
////            local = mDPM.getPasswordMinimumLowerCase(mDeviceAdminSample);
////            global = mDPM.getPasswordMinimumLowerCase(null);
////            mMinLowerCase.setSummary(localGlobalSummary(local, global));
////            local = mDPM.getPasswordMinimumUpperCase(mDeviceAdminSample);
////            global = mDPM.getPasswordMinimumUpperCase(null);
////            mMinUpperCase.setSummary(localGlobalSummary(local, global));
////            local = mDPM.getPasswordMinimumSymbols(mDeviceAdminSample);
////            global = mDPM.getPasswordMinimumSymbols(null);
////            mMinSymbols.setSummary(localGlobalSummary(local, global));
////            local = mDPM.getPasswordMinimumNonLetter(mDeviceAdminSample);
////            global = mDPM.getPasswordMinimumNonLetter(null);
////            mMinNonLetter.setSummary(localGlobalSummary(local, global));
//        }


       @Override
       public boolean onPreferenceChange(Preference preference, Object newValue) {
//             if (super.onPreferenceChange(preference, newValue)) {
//                 return true;
//             }
//             String valueString = (String)newValue;
//             if (TextUtils.isEmpty(valueString)) {
//                 return false;
//             }
//             int value = 0;
//             try {
//                 value = Integer.parseInt(valueString);
//             } catch (NumberFormatException nfe) {
//                 String warning = mActivity.getString(R.string.number_format_warning, valueString);
//                 Toast.makeText(mActivity, warning, Toast.LENGTH_SHORT).show();
//             }
//             if (preference == mPasswordQuality) {
//                 mDPM.setPasswordQuality(mDeviceAdminSample, value);
//             } else if (preference == mMinLength) {
//                 mDPM.setPasswordMinimumLength(mDeviceAdminSample, value);
//             } else if (preference == mMinLetters) {
//                 mDPM.setPasswordMinimumLetters(mDeviceAdminSample, value);
//             } else if (preference == mMinNumeric) {
//                 mDPM.setPasswordMinimumNumeric(mDeviceAdminSample, value);
//             } else if (preference == mMinLowerCase) {
//                 mDPM.setPasswordMinimumLowerCase(mDeviceAdminSample, value);
//             } else if (preference == mMinUpperCase) {
//                 mDPM.setPasswordMinimumUpperCase(mDeviceAdminSample, value);
//             } else if (preference == mMinSymbols) {
//                 mDPM.setPasswordMinimumSymbols(mDeviceAdminSample, value);
//             } else if (preference == mMinNonLetter) {
//                 mDPM.setPasswordMinimumNonLetter(mDeviceAdminSample, value);
//             }
//             reloadSummaries();
           return true;
       }


   }
}


/*
* DatabaseHelper.java - performs DB operations
*/
package sp.remote.database;


import sp.remote.app.MyLog;
import sp.remote.model.K;
import sp.server.Outlet;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
```

```
public class DatabaseHelper {
        //SOCKETS{ _id INT NOT NULL | _name TEXT NOT NULL | _type INT NOT NULL | _status INT NOT NULL
| _roomid INT | _ip TEXT NOT NULL }



        public static Cursor getAllEntries(SQLiteDatabase db, String name) {
                Cursor cur = db.query(name,
                null, null, null, null, null, null);
                MyLog.d("justin", "Count in Helper: " + cur.getCount());


                return cur;


        }


        public static int updateLight(SQLiteDatabase db, String name, Outlet outlet) {
                ContentValues cv=new ContentValues();
                cv.put(K.DB_COL_STATUS, outlet.getkStatus());
                return db.update(name, cv, K.DB_COL_ID+"=?", new String[] {
String.valueOf(outlet.getkId()) } );
        }
}


/*
* DatabaseInitializer.java - creates DB on local android client
*/
package sp.remote.database;


import sp.remote.app.MyLog;
import android.content.Context;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;


public class DatabaseInitializer extends SQLiteOpenHelper {


        private static final int DATABASE_VERSION = 1;
        public static final String TABLE_NAME = "SOCKETS";
        private static final String CREATE_TABLE =
                        "CREATE TABLE " + TABLE_NAME +
                        " (_id INTEGER PRIMARY KEY ASC, " +
                        "_type TEXT NOT NULL, " +
                        "_name TEXT NOT NULL, " +
                        "_status INTEGER, " +
                        "_dimVal INTEGER, " +
                        "_ip TEXT NOT NULL, " +
                        "_mac TEXT NOT NULL);";


        public DatabaseInitializer(Context context) {
                super(context, TABLE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
                MyLog.d("justin", CREATE_TABLE);
                try {
                        db.execSQL(CREATE_TABLE);
                }
                catch (SQLException ex) {
                        ex.printStackTrace();
                }
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        }


}


/*
* K.java - holds constants
*/
package sp.remote.model;
```

```
public abstract class K {

        public static final int POWER_TYPE_SOCKET = 123;
        public static final int POWER_TYPE_SWITCH = 543;


        public static final int STATUS_ON = 1;
        public static final int STATUS_OFF = 0;
        public static final int STATUS_DIMMED = 77;


        public static final String STATUS_ON_STRING = "ON";
        public static final String STATUS_OFF_STRING = "OFF";


        public static final String DB_COL_STATUS = "_status";
        public static final String DB_COL_TYPE = "_type";
        public static final String DB_COL_NAME = "_name";
        public static final String DB_COL_ID = "_id";
        public static final String DB_COL_IP = "_ip";


        public static final String MESSENGER = "messenger_key";


        public static final String SERVICE_ACTION_KEY = "action_key";
        public static final int ACTION_REFRESH_DB = 43;
        public static final int ACTION_CONNECT_SERVER = 44;
        public static final int ACTION_TURN_ON = 45;
        public static final int ACTION_TURN_OFF = 46;
        public static final int ACTION_ACK = 47;
        public static final int ACTION_DELETE_DB = 48;
        public static final int ACTION_DISCONNECT_SERVER = 55;


        public static final String SERVICE_ID_KEY = "id_key";
        public static final String SERVICE_DIM_VAL_KEY = "dimmer_key";



        public static final String ACTIVITY_RESPONSE_KEY = "gui_key";
        public static final int RESPONSE_REFRESH_DB = 91;

}
```

APPENDIX D: ANALYSIS OF SENIOR PROJECT DESIGN

Project Title: LyFi

Student's Name:

Kianoosh Salami (EE), Justin Cotton (CPE), Elush Shirazpour (CPE)

Advisor's Name: Bryan Mealy **Advisor's Initials**: **Date**:

• Summary of Functional Requirements

LyFi allows a mobile application to control a light's state (on/off) and dimmer value remotely over WiFi. LyFi's hardware can be modified to control other appliances in a home.

• Primary Constraints

Initially we were unsure how the Server and Client would seamlessly connect. We found we could have our Server listen for a UDP broadcast packet with a secret key. Once the key was found the Server now knows the Client's IP information and can forward its personal IP to the client. This allows the Server and Client to connect dynamically without any user configuration.

Our biggest challenge was optimizing our firmware on hardware with very limited SRAM, as FreeRTOS uses a significant amount on its own. We experienced many crashes due to our Stack and Heap colliding. We solved this by moving our large buffers from SRAM to EEPROM. Although technically slower, the extra latency is unnoticeable to the end user. If this project would to be redesigned, we would switch to a more powerful ARM microprocessor.

• Economic

The economic factors led by the LyFi will result in human capital and manufactured capital. LyFi's PCB requires copper and silicon which would result in the increase of sales in copper and silicon. The project costs around 95 dollars and the team members pay for the cost for the parts and to assembly. The team does assembly as well so that saves on labor costs. Cost to create the device is around 95 dollars so profit will be made once the LyFi product is sold at retailer's shops. The project earns around 6 dollars per product if sold at 100 dollars per product. The estimated value of the product at the start of the project was 150 dollars, but we have it now to 95. We do not plan to market this product after this project. Below is a bill of materials for our LyFi product.

BILL OF MATERIALS (PROTOTYPE / MASS COST)

Part Name	Part Number	QTY	Cost Per Part	Prototype Cost	Mass Cost
Atmega328P	Atmega328P TQFP	1	$4.30	$4.30	$1.70
GainSpan WiFi	GS1011MEP	1	$35.00	$35.00	$25.00
8MHZ Crystal	CTX1054	1	$0.46	$0.46	$0.27
PCB Assembly	LyFi Embedded	1	$33.00	$33.00	$5.00
Jumper	Jumper Shorting .1"	3	$0.05	$0.15	$0.05
Header 36Pack	Molex 10-89-7181	1	$1.69	$1.69	$0.57
22pf Capacitor	22pf 0805 Cap	2	$0.04	$0.08	$0.04
0.01uf Capacitor	0.01uf 0805 Cap	2	$0.04	$0.08	$0.01
10K Resistor	10k 0805 Resistor	3	$0.17	$0.51	$0.01
5.6ohm Resistor	5.6 0805 Resistor	1	$0.15	$0.15	$0.01
56ohm Resistor	56 0805 Resistor	1	$0.15	$0.15	$0.01
10uf Capacitor	10uf 0805 Resistor	1	$0.15	$0.15	$0.01
0.1uf Capacitor	0.1uf 0805 Cap	1	$0.06	$0.06	$0.01
Push Button Switch	Omron B3F	2	$0.35	$0.70	$0.36
Standoff Screw	1/4" 4-40	4	$0.20	$0.80	$0.16
Standoff Stand	Keystone 3481	4	$0.45	$1.80	$0.80
Encoder	Rotary Encoder	1	$2.95	$2.95	$1.11
AC/DC Switching Regulator	RAC01-3.3SC	1	$14.52	$14.52	$8.24
Triac	BT136	1	$0.60	$0.60	$0.18
Diode Rectifier	DF15005S	1	$0.17	$0.17	$0.12
Optocoupoler	4N25 SMD	1	$0.38	$0.38	$0.17
Triac Driving Optoiso	MOC3021 SMD	1	$0.80	$0.80	$0.24
330uF Capcitor	330uF Electrolytic	1	$0.25	$0.25	$0.04
10k Resistor	10k 0805 Resistor	1	$0.17	$0.17	$0.01
47k Resistor	47k .33W 0805	2	$0.27	$0.54	$0.02
1k Resistor	1k 0805 Resistor	1	$0.15	$0.15	$0.01
470ohm Resistor	470 0805 Resistor	1	$0.15	$0.15	$0.01
Standoff Screw	1/4" 4-40	5	$0.20	$1.00	$0.20
Screw Terminal	PRT-08432	2	$0.95	$1.90	$1.52
PCB	LyFi HiV	1	$33.00	$33.00	$5.00
	Var Shipping	1	$10	$10.00	$0
Total				$145.66	$50.88
				Dif	-$94.79

• If manufactured on a commercial basis:

There are an estimated 1billion smartphones on the market. Each user is potentially a customer. If we sell to 0.1% of this market, that is 1million units sold.

Cost Prototype: $145.66 Estimated Mass Production Price $50.88

If 1Million units are sold at $75 that is a revenue of $75Million and profit of $24.12Million.

A new WiFi Chipset would significantly save cost. If mass-producing, a cheaper ARM processor with WiFi would be used. This requires a new system design, however cost can be decreased ~$15.

• **Environmental**

LyFi's devices are made mostly of silicone, plastic and copper. If LyFi was to go into mass production, materials such as rubber, solder, polyurethane, acrylic, or epoxy will need to be used. This can impact the environment since all those materials come from the environment. Since the biggest element of the LyFi is silicone, which is sand, the environment will not be hurt due to mass production. The product will not impact any animals at all since the product is not too demanding for environmental products. Since the product is so small, it will not have a major impact on the environment or any animals.

• **Manufacturability**

Manufacturability should not be challenging as the device uses off the self parts and is easily assembled.

• **Sustainability**

There are no moving parts; therefore the device should sustain its quality throughout the lifetime of the product. The most negative attribute related sustainability is the device consumes power (roughly 2mW). Although the power consumption is minimal, in a grand scale it could potentially cause an impact. The dimmer feature does however utilize improved dimming technology, and consequently can save a significant amount of power if the user chooses to keep lights under full brightness.

• **Ethical**

There is a potential for a malicious hacker to gain control of the device and control nodes remotely.

• **Health and Safety**

Installation can be dangerous, as the user would be required to deal with household AC voltages. Due to the convenience of controlling devices remotely, users overtime may become lazier.

• **Social and Political**

This product can potentially change a behavior of a person. No longer will a user be required to have physical access to a device controlled by LyFi.

• **Development**

Firmware was implemented using AVR Studio and a STK600 development board.
Debugging through the serial port used an FTDI USART to USB converter in junction with CoolTerm for OSX.
Schematic and PCB layout was done with Eagle CAD.
The Server and Android Application were developed using the Eclipse IDE with OSX.
Eclipse utilized the Android SDK for mobile development.