Iowa State University

Efficient Home Lighting
Final Report

May 1637
Advisor: Gary Tuttle
Client: Gary Tuttle

Flavia Cavalcanti, Ryan Marion, Alex Rinehart,
John Stabenow, Mitchell Wheaton, David Wiest

TABLE OF CONTENTS

PROJECT DESCRIPTION **3**

SYSTEM LEVEL DESIGN **3**

SYSTEM REQUIREMENTS 3
FUNCTIONAL DECOMPOSITION 3
SYSTEM ANALYSIS 3

PROJECT DETAILS **4**

I/O SPECIFICATION 4
INTERFACE SPECIFICATIONS 4
SOFTWARE/EMBEDDED SPECIFICATION 5
Embedded Microcontroller System *5*
The following algorithmic flowchart details the process in which the microcontroller makes decisions and interfaces with the user: *6*
iOS Mobile Application *7*
Android Mobile Application *7*
HARDWARE SPECIFICATION 7
Switch-Mode Power Supply *7*
Embedded Microcontroller System *10*
LED Bulb *12*

TESTING PROCESS AND RESULTS **15**

SYSTEM LEVEL TESTING 15
SWITCH-MODE POWER SUPPLY 17
EMBEDDED MICROCONTROLLER SYSTEM 18
General Functionality *18*
Bluetooth Signal Range Testing *19*
Power consumption *19*
MOBILE APPLICATION 19
iOS *19*
Android *20*
LED BULB 20

APPENDIX I: OPERATION MANUAL **A1**

Embedded Systems Maintenance *A2*

APPENDIX II: ALTERNATIVE DESIGN VERSIONS **B1**

MOBILE APPLICATION B1
MICROCONTROLLER/BLUETOOTH B1
POWER SUPPLY B1

APPENDIX III: OTHER CONSIDERATIONS **C1**

APPENDIX IV: CODE **D1**

IOS MOBILE APPLICATION D1
ANDROID MOBILE APPLICATION D22
EMBEDDED MICROCONTROLLER SYSTEM D42
MATLAB FOR SMPS FEEDBACK NETWORK D51

Project Description

Modern advances in LED technologies has led to large efficiency gains in the home lighting industry; however, the cost per LED is still much higher than alternatives. An LED bulb consists of a power supply and several LEDs. Each power supply consists of components like large electrolytic capacitors that can fail easily. The lifespan of LED bulbs can be increased by moving the power supply away from the bulb. The purpose of our project is to design a system including LED bulbs, switch-mode power supplies and a lighting control system that will improve upon these inefficiencies/costs. This project covers the whole spectrum of computer engineering/electrical engineering topics and skills.

The problem stems from the close proximity of the LEDs to the power supply as shown in ***Figure 1***. Our solution implements the switch-mode power supply (SMPS) into the switch box (***Figure 2***). This has several advantages, such as: improved lifespan due to reduced heat stress on the SMPS, ease of integration of smart home features, and reduced bulb cost. The ability of adding a microcontroller to control the lights is due to the central location for a processing unit, as well as easy access to control the lights. With only a single SMPS that can power several bulbs, the number of components per bulb decreases dramatically compared to standard LED bulbs.

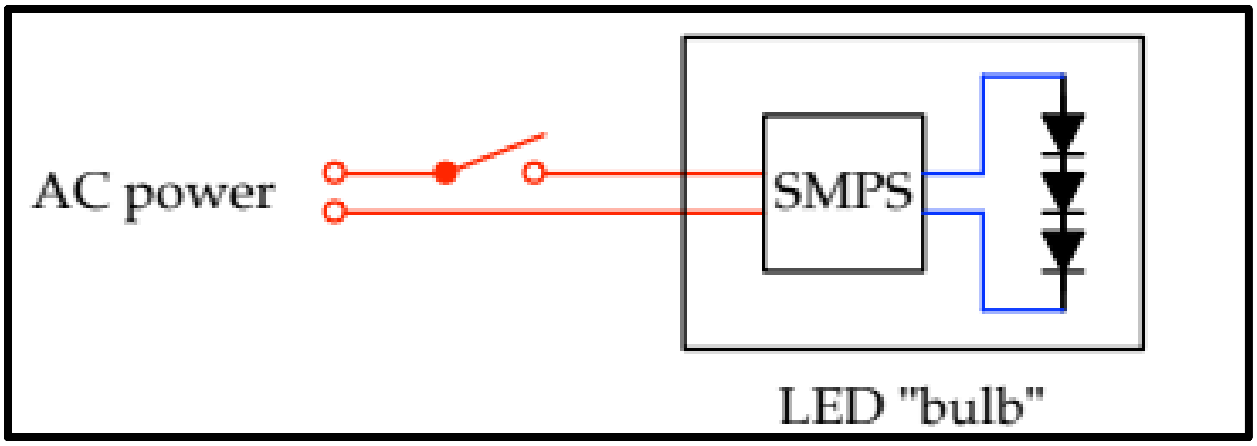


Figure 1. *Existing LED bulbs utilize SMPS inside of bulb which increases cost and decreases lifespan of the bulb.*

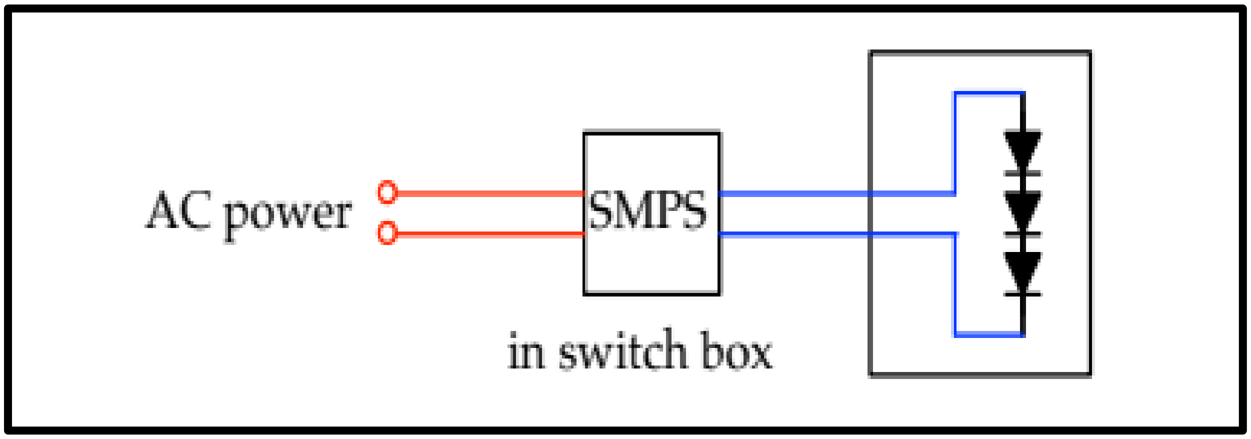


Figure 2. *Utilize the switch box to house a single SMPS that can power multiple LED bulbs, reducing on component costs and increasing lifespan.*

The end goal of our project is to implement a home lighting system that has a single SMPS and can be easily integrated into an existing home. The SMPS also has smart capability with Bluetooth and the ability to connect to a mobile app.

System Level Design

System Requirements

The efficient home lighting project requires several components including software and hardware. One of the most important components of the project is designing a switching power supply that will efficiently convert an AC voltage into a regulated DC voltage. Another component is an embedded control system including a microcontroller and Bluetooth module. The size of the power supply and embedded system are restricted by the size of a light switch box. Another important component of the project is the LED bulb that will consist of several LEDs, a mechanical enclosure, and a protection circuit. The last component of the project is a mobile application that can send commands to the embedded controller to control the lights from a person’s phone. **Table 1** gives a table of requirements for each sub-system.

System Specifications				
	Switched Mode Power Supply	Microcontroller/Bluetooth	LED Bulb	Mobile App
1	Use Existing 120V House Wire	Program with ICSP Header	Equivalent of 60W Incandescant	iOS and Android App
2	Properly Isolated	Communicate with app via BLE	Homebrew PCB Fab	Control Separate Lights
3	18V, 700mA Output	Incorporate a Real-Time Clock	Current Limiting Circuit Design	All Lights On/Off
4	5V, 100mA Output	Hardware Switch and Dimmer	Light Spectrum Testing	Get Light State from Microcontroller
5	Minimal Size	Periodic Power Usage Measurement		Display Power Usage
6		Keep Track of Dimming State		

Table 1. *System Specifications.*

Functional Decomposition

The functional decomposition of this project can be broken into two inputs. The first input is the user sending commands via a mobile application or pressing hardware switches and the second is the AC mains power. The AC mains is what will power the SMPS, microcontroller, Bluetooth, and LED bulbs. It will be at 120 Vrms AC. The user input will be from the iPhone app, as well as the light switches on the wall.

System Analysis

As a whole, the system will have a single power supply that will power multiple banks of LEDs. This allows the power supply to be optimized for efficiency without the normal constraints of placing the SMPS in a light bulb. The existing AC wires can then be rewired in the switch box to function as the DC lines for the bulbs. This idea will allow existing structures to rewire the lights without pulling new wire, which can be difficult and expensive. The only parts that will need replaced are the switch box and light bulb in order to update an existing house to our bulbs.

Project Details

I/O Specification

The efficient home lighting system is fairly simple in terms of input and output specifications. The two main inputs into the system will be the 120V AC voltage from the home and the user input from the mobile application. The output from the system will be light from the LEDs. A user will be able to control lights with their smart device by using an intuitive user interface on a mobile application.

Interface Specifications

Each component in the system needs a well-defined interface in order for the system to remain modular. Modularity is necessary to allow an efficient design and revision process. Below is a short description of each component's interfaces. The modular specification is shown in the block diagram in ***Figure 3***.

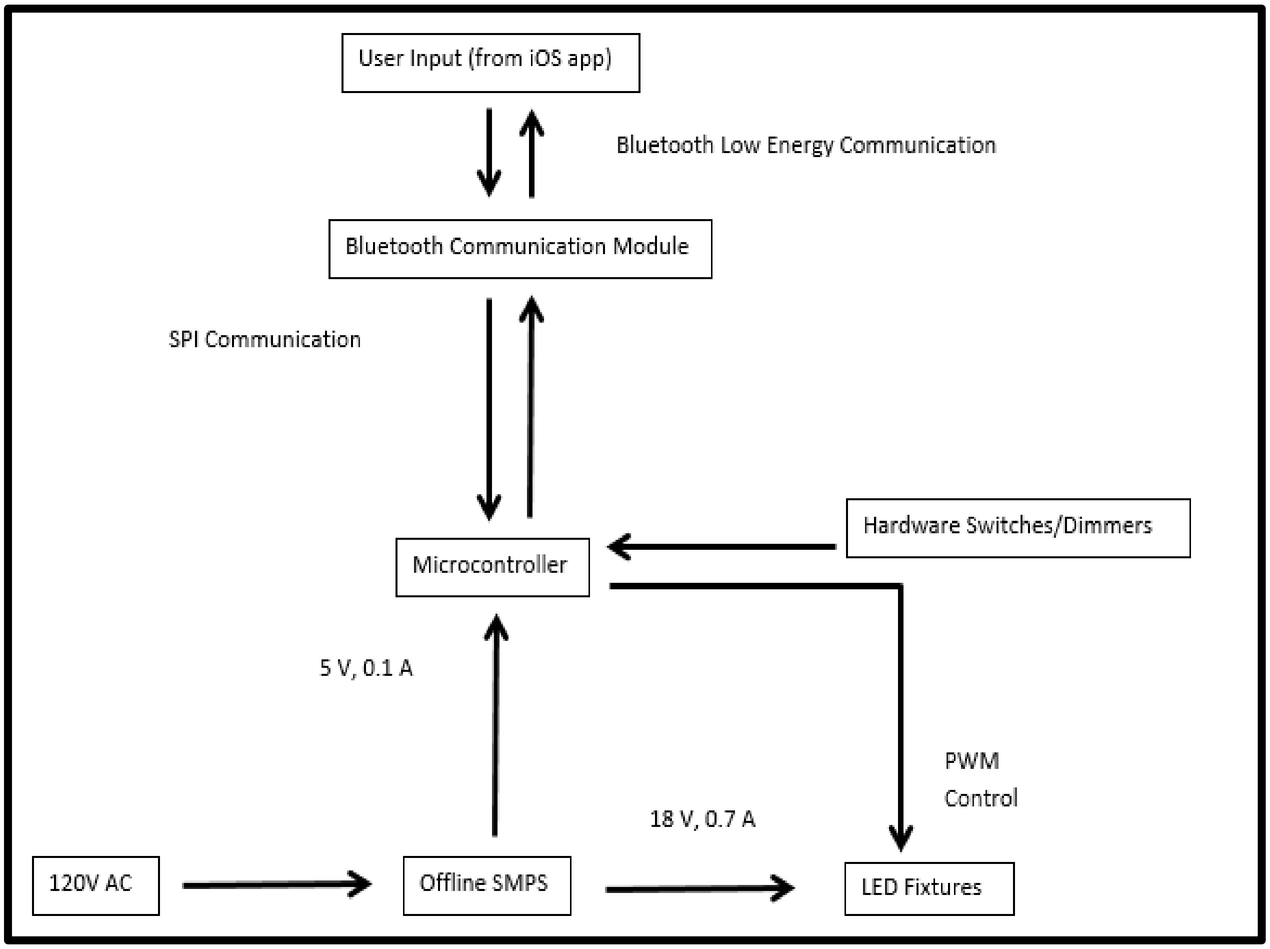

User Input (from iOS app)
Bluetooth Low Energy Communication
Bluetooth Communication Module
SPI Communication
Hardware Switches/Dimmers
Microcontroller
5 V, 0.1 A
PWM Control
18 V, 0.7 A
120V AC
Offline SMPS
LED Fixtures

Figure 3. *System Block Diagram.*

Software/Embedded Specification

Embedded Microcontroller System

Microcontroller

The standard Arduino C++ libraries are used to control the ATMega328 microcontroller. These libraries allow us to read analog voltages, digital states, and manage serial objects. Using the open-source Arduino libraries, we were able to focus on implementation details without recreating existing functionality.

Bluetooth

The firmware includes a Nordic Semiconductor nRF8001 C++ library implementing Bluetooth 4.0 over UART. Nordic Semiconductor library creates an application controller interface that allows the microcontroller to use the nRF8001 over a serial connection. The application controller interface can exchange system and event information with any connected device. The types of information include system commands, system events, data commands, and data events. This information is used by the microcontroller to detect whether a device is connected and then receive or send information to the paired device.

Real-time Clock

The Maxim DS1307 real-time clock is controlled using an open-source library that allows an atmega328 to communicate with the chip via I2C serial communication. The RTC library uses the wire Arduino library to implement the I2C serial communication. The library also includes objects that can be created to keep track of the time and differences between times.

The block diagram of the system for the microcontroller is detailed below. The microcontroller communicates with the Bluetooth module via SPI configured to serve as a UART interface. This is seen below in ***Figure 4****.*

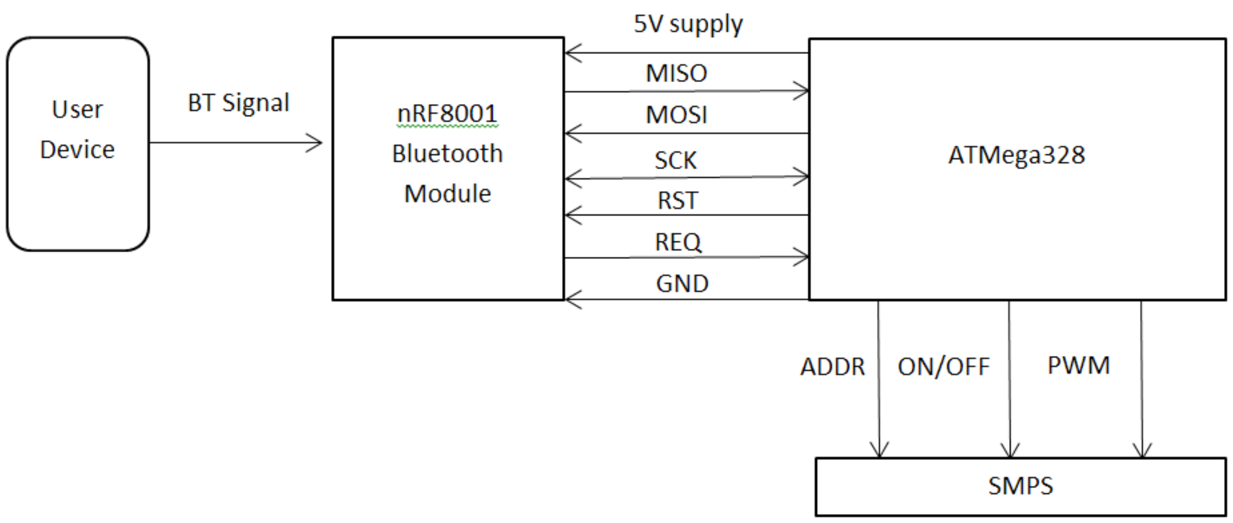
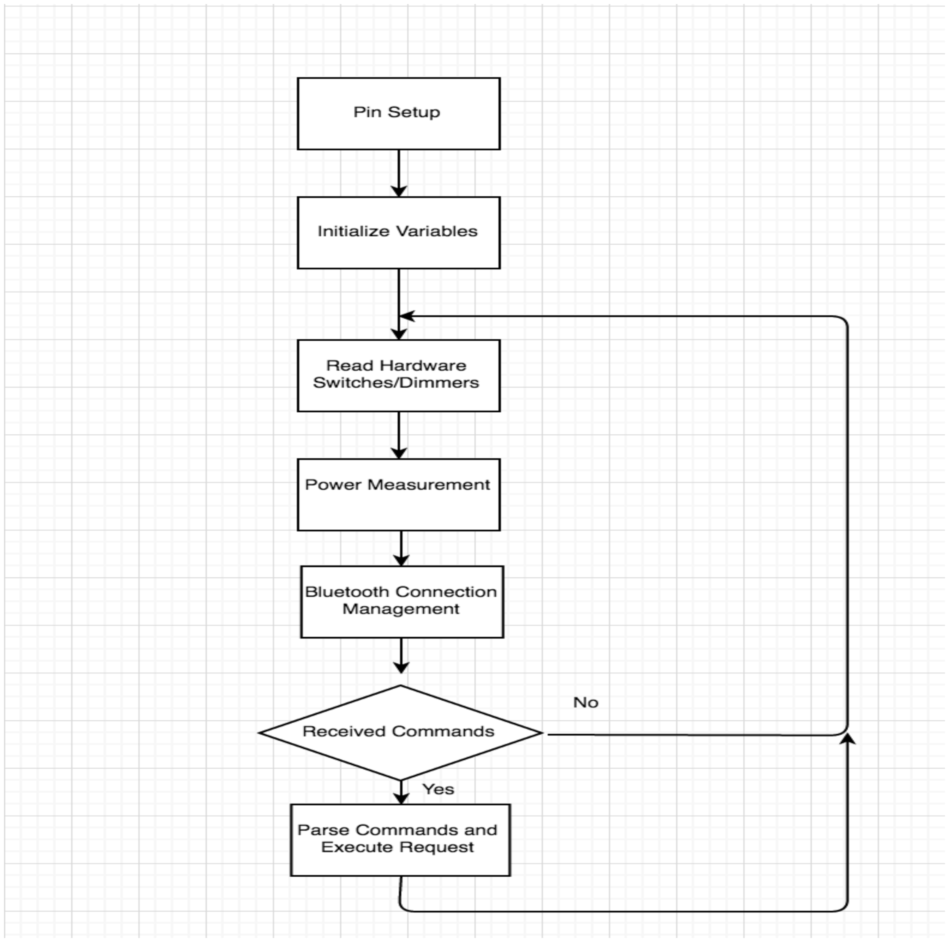


Figure 4. *Microcontroller Block Diagram*

The following algorithmic flowchart details the process in which the microcontroller makes decisions and interfaces with the user:


Pin Setup
Initialize Variables
Read Hardware Switches/Dimmers
Power Measurement
Bluetooth Connection Management
Received Commands
No
Yes
Parse Commands and Execute Request

Flowchart 1: Embedded System

iOS Mobile Application

The iOS application will run on iOS 6 or higher. The basic UI must allow users to connect to a given microcontroller, to modify a selected light, and to save their preferences. When forming a connection to the microcontroller, the application shall poll the microcontroller for the state of the lights, and adjust the software switches accordingly. The application shall provide the user with the ability to adjust the brightness of each light, as well as the on/off state of each light. iOS applications use MVC design patterns.

Android Mobile Application

The Android application shall run on API 18 and higher. The look of the application shall mimic the look of the iOS application. When forming a connection to the microcontroller, the application shall poll the microcontroller for the state of the lights, and adjust the software switches accordingly. The application shall provide the user with the ability to adjust the brightness of each light, as well as the on/off state of each light. Android applications utilize an activity stack to pass data between screens.

Hardware Specification

Switch-Mode Power Supply

The switch mode power supply (SMPS) should take in the existing AC voltage found in all buildings, and regulate down to the voltage and current needed for a LED string (18 V DC and 700mA in this case). The SMPS should also have an additional 5V output to supply the microcontroller and Bluetooth module.

Originally, a flyback converter was chosen for proper electrical isolation and efficiency. Basic flyback converter operation can be seen in **Figure 5.**

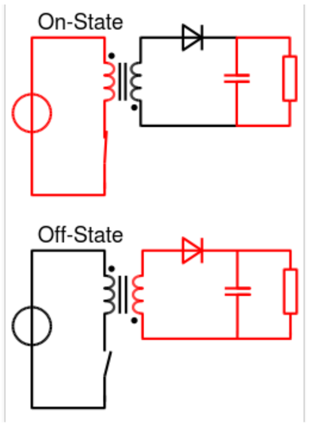


Figure 5. *Basic Flyback Converter Operation*

The input to the system is high voltage DC that is generated by rectified wall voltage. While the switch (a high voltage NMOS) is closed, energy is stored magnetically in the transformer. The dot negative connotation of the transformer indicates that there is a 180 degree phase shift between the primary and secondary. Energy is then dumped into the secondary side while the switch is open. This isolation also extends to the feedback network, significantly complicating the design of the SMPS. To design the feedback network, MATLAB was used to simulate the transfer functions of the switching controller, the transformer, and the output filter as well as the feedback network. This code can be seen in **Appendix IV**, as well as the formula for the transfer functions inside the code. A PCB was also designed in NI Ultiboard, and the schematic used to create the board footprints can be seen below in **Figure 6**.

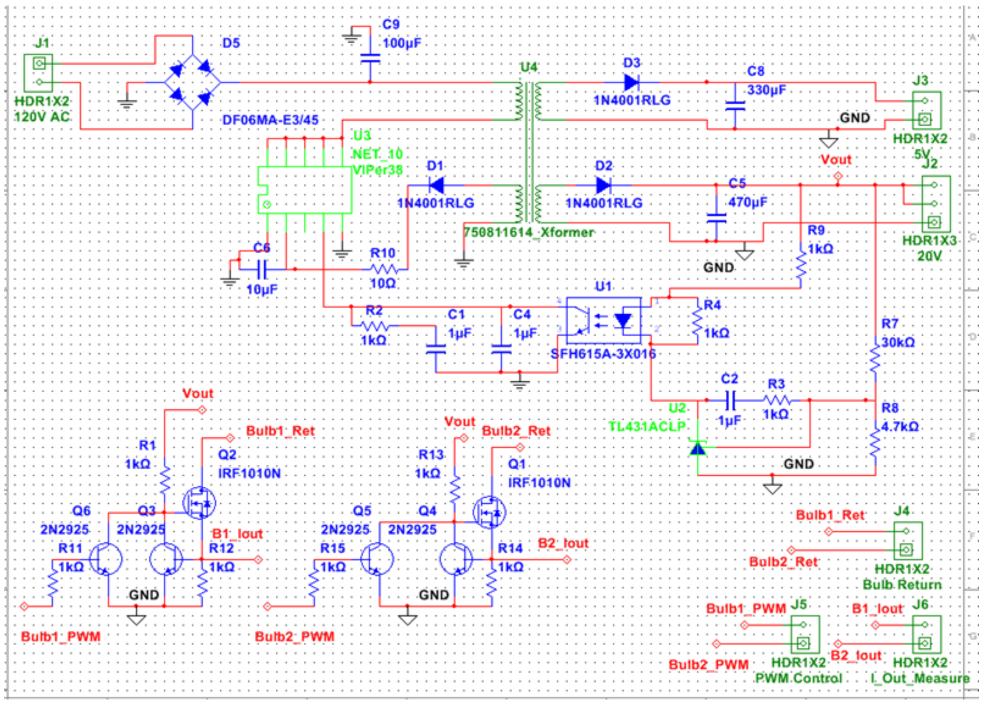


Figure 6.*Schematic of Flyback Converter with 2 Outputs and Current Limiting Circuit*

This circuit was then used to create the PCB, which measured less than 2x3 inches. This PCB can be seen in ***Figure 7.***

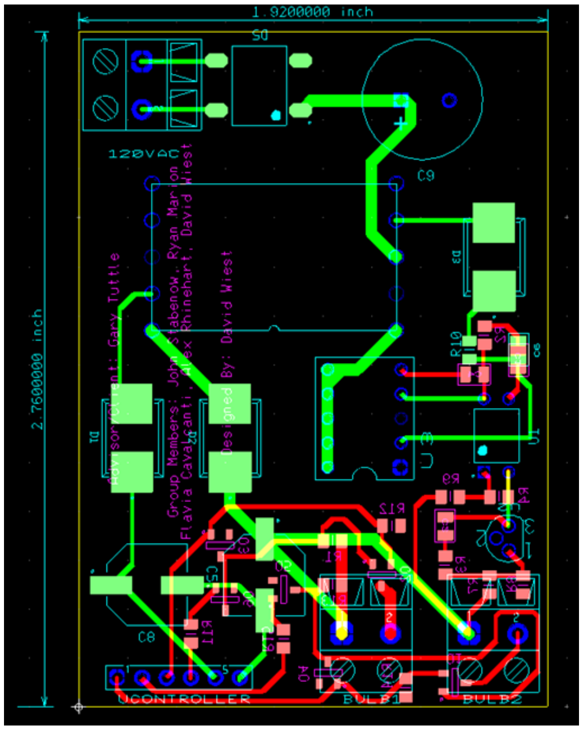


Figure 7. *PCB Layout for Flyback Converter*

Unfortunately, time constraints kept the design from being fully implemented, and the design will be mostly completed at the end of the semester. An interim design consisting of a step down transformer and a buck converter was used to meet the power requirements. A second buck converter was then used to step down from 18 to 5V to power the microcontroller. **Figure 8** shows the fabricated regulator, as well as the step down transformer.

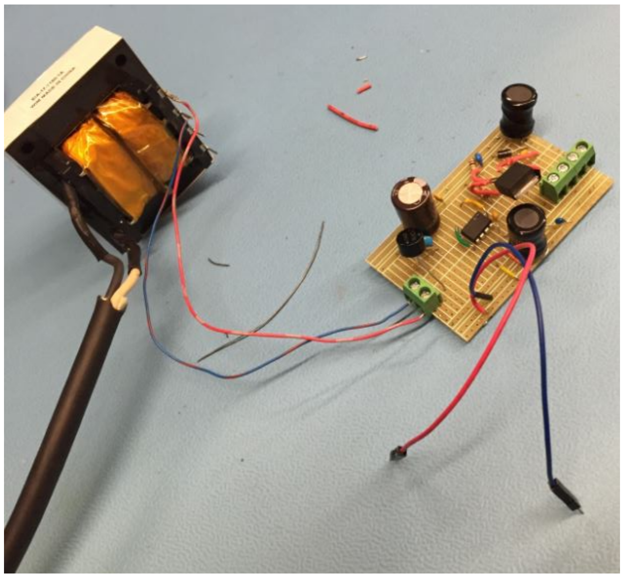
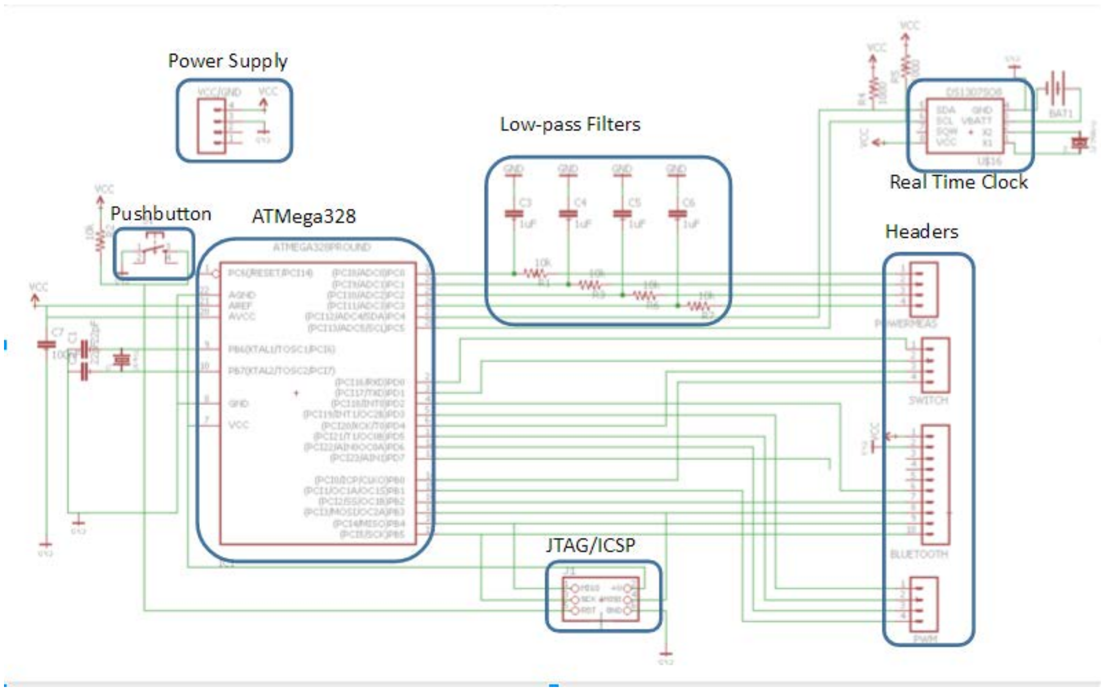


Figure 8. *Soldered PCB of Interim SMPS*

Embedded Microcontroller System

This section is an outline of the embedded system's interfaces including inputs and outputs. The embedded system should be powered with a regulated 5V supplied by the SMPS. The system's outputs include PWM signals to control the state of the light and status information sent via Bluetooth 4.0. The PWM signal is used in a current control circuit to adjust the light bulb's brightness. The embedded system is also reading the state of hardware switches and adjusting the lights accordingly. The on/off switches are being interpreted as a digital value and the dimmer switches are being read in as an 8-bit analog value. The system is also calculating the power consumption of each individual light bulb. The system reads in a DC voltage across a resistor in a current control circuit and calculates the power being consumed using Ohm's law.

The microcontroller team decided to create two versions of PCBs for the project. One version the PCB has a through-hole atmega328 microcontroller while the other version has a QFN surface-mount version of the atmega328 microcontroller. This allows us to reduce the amount of risk involved with manufacturing the PCBs. Each PCB is about the same dimensions in size. The following figure is the schematic for each design:



Below is an image of the through-hole version of the PCB.

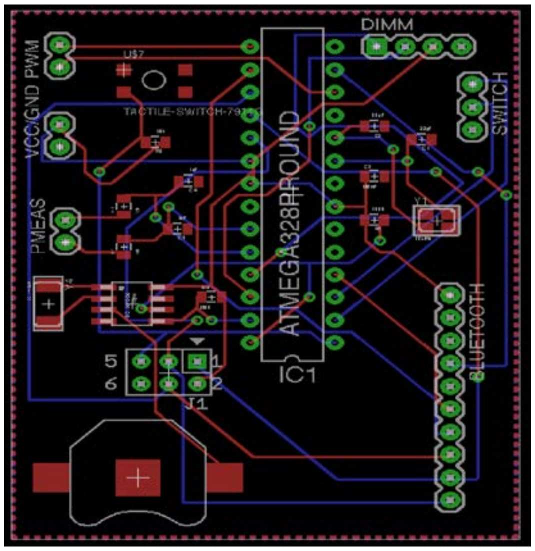


Figure 9. Through-hole PCB

The next figure is an image of the surface-mount PCB. You can see the size of the microcontroller decreased tremendously. The size of the PCB could be reduced even more if the power supply was on the same PCB. This would reduce the amount of headers needed.

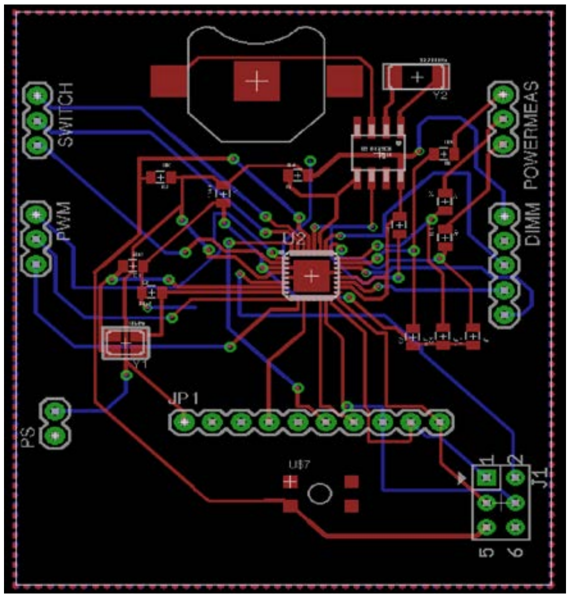


Figure 10. Surface Mount PCB

Below are images of the final PCB with all of the components soldered into place.

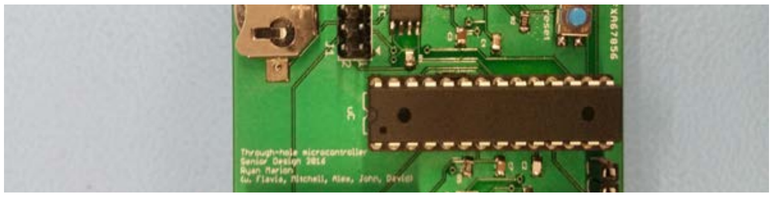


Figure 11. Through-hole PCB

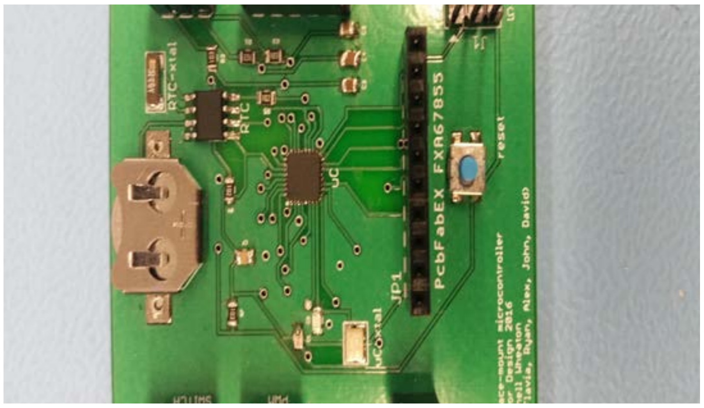


Figure 12. Surface Mount PCB

LED Bulb

The LED Bulb is designed to be a replacement for a generic 60W 120V bulb. Since the power lines to the housing will now be DC, it is important to keep the polarity correct.

Our power supply outputs 18V DC, which is what our bulbs need to run off of. A typical 60W bulb has a lumen output of 750 lumens to 900 lumens. The color correlated temperature is the associated spectrum for a black body radiator at a given temperature. One important consideration is also the efficiency, we want our bulb to be greater than 100 lumen/W. Using these constraints we can give an outline of our specifications.

- 18V DC
- Color Correlated Temperature of 3000K
- 60W Equivalent (750-900) lumens
- Power Consumption Less than 12W
- Efficient (>100lm/W)

After looking at the given constraints, we were able to decide on a given LED to buy. We went with an LED from Seoul Semiconductor (SZ5-M1-WW-C8-V1/V2-GA) outlined in the table below. In order to complete a given bulb, six LEDS are needed per PCB.

CCT (K)	Lm/W	Flux(lm)	If(mA)	Vf(V)	CRI	#Per Bulb
3050	120	124	350	2.95	80	6

Table 2. *Seoul Semiconductor LED Parameters.*

With the LEDs specified, the next step was in designing a PCB layout. The PCB design is shown below in **Figure 13*.*** The six LEDs are arranged in a circular pattern to distribute the light evenly. The fabrication of the PCB boards is outlined in **Appendix III**.

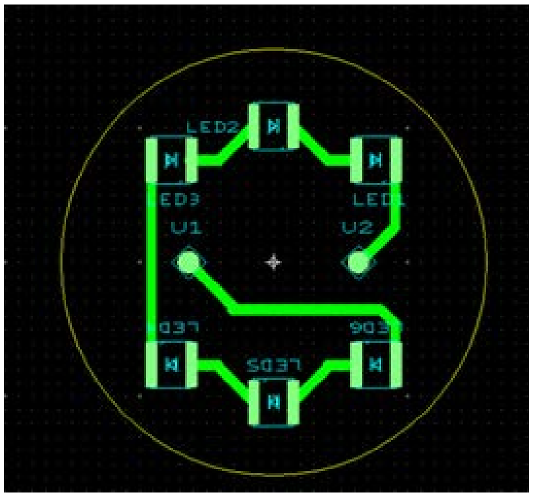


Figure 13. *Light Bulb PCB Layout*

Part of designing the LED bulbs is in designing a current limiting circuit. The schematic for this circuit is outlined in ***Figure 14***. The function of this circuit is to limit the current to each bulb to 350mA and to allow the microcontroller to supply a PWM signal safely to the bulbs. The current is set by the resistor across the base emitter junction (.7V drop). The current limiting circuit is on the power supply board and is then sent out to the LEDS.

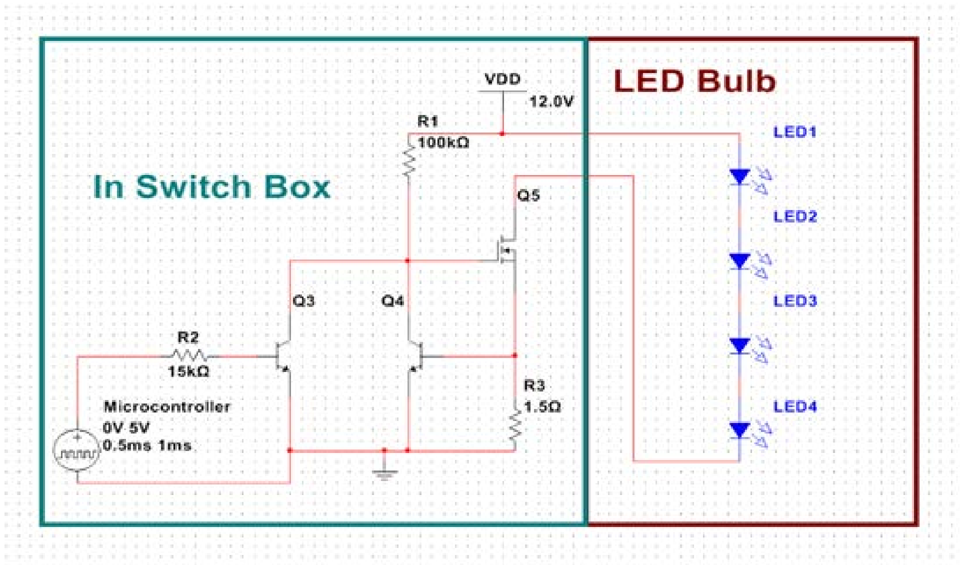


Figure 14. *LED and Current Limiting Circuit*

Testing Process and Results

System Level Testing

In order to verify that the overall system functions correctly, the whole project was assembled into a demo fixture (***Figure 15)***. The Microcontroller board and power supply were bundled together into a stack to allow them to easily fit into the test fixture (***Figure 16).*** The switches and dimmers were then wired to work with the microcontroller (***Figure 17).***

With all of the system together, we were able to verify correct operation of all of the components. The switches, dimmers, and app connected and worked with the microcontroller to dim and turn the lights on and off. The demo box layout is very similar to what would be in a typical household that would be retrofitted with our system.

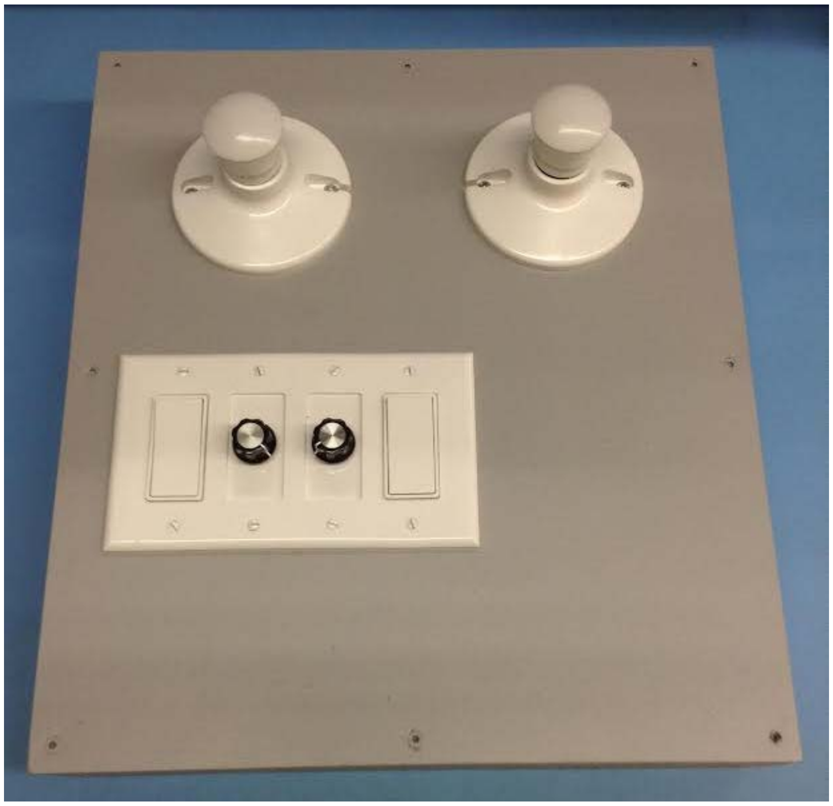


Figure 15. *Demo fixture utilized to test system operation.*

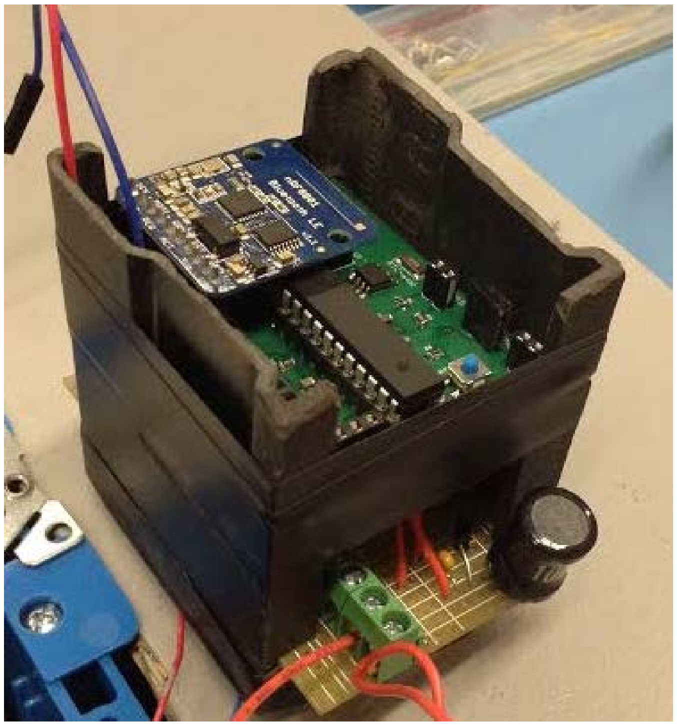


Figure 16. *Microcontroller and power supply stacked on top of each other to fit inside switch box.*

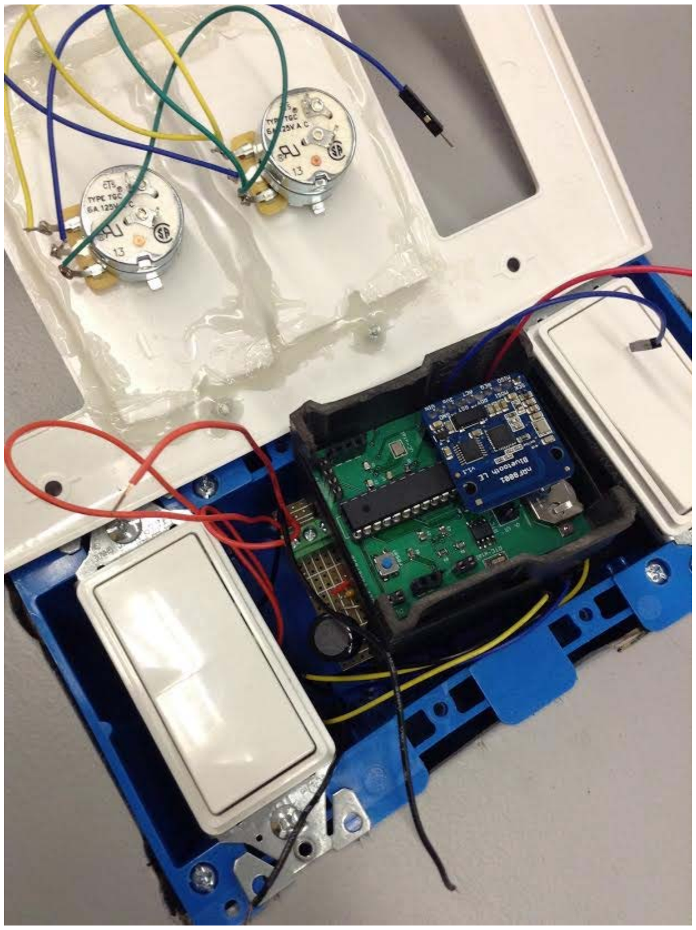


Figure 17. *Light switches and dimmers attached to microcontroller.*

Switch-Mode Power Supply

Testing the SMPS is a straightforward process with two common metrics - efficiency and load regulation. To test efficiency, the input power will be measured using a sense resistor and the output power will be the sum of the power delivered to the LED Strings and the Microcontroller/Bluetooth module. The maximum output power is

$$Pout = 18 * 0.75 + 5 * 0.1 = 14\,W$$

Input power will be measured by

$$Pin = 170 * Measured\ Iin\ W$$

Efficiency is then calculated as

$$\eta = \frac{Pin}{Pout}$$

Load regulation will be tested by measuring the output voltage under no load conditions (lights off) and then measuring the output voltage under full load conditions (both lights on). The load regulation can then be expressed as

$$\text{Load Regulation} = \frac{Vout(No\ Load)}{Vout(Full\ Load)} * 100$$

To test load regulation. The circuit in **Figure 18** was used to draw current. In this circuit, current is controlled by setting a voltage at the inverting terminal of an amplifier, and using the non-inverting terminal to set the voltage across a 1 Ohm resistor as the load. The initial buffer and voltage divider is used to set the maximum current flow.

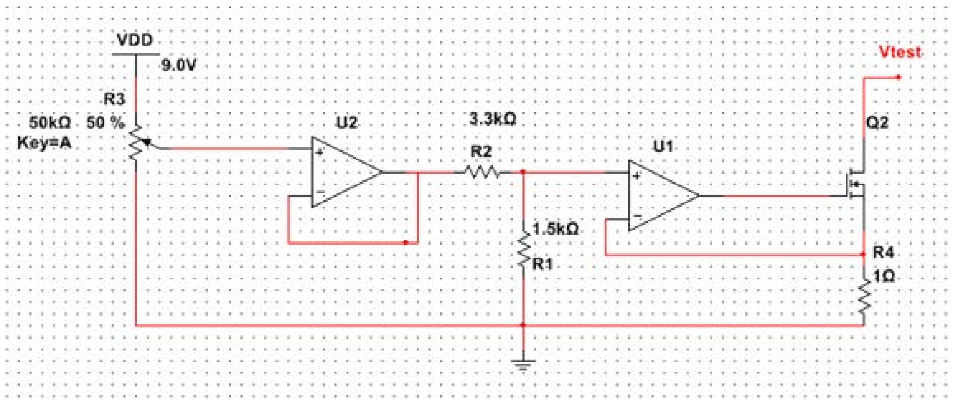


Figure 18. *Circuit used to test load regulation.*

Embedded Microcontroller System

Before testing the functionality of the board, we performed continuity testing on the PCB to verify there were no shorts where they shouldn't be. After we eliminated all shorts on the printed circuit board we, applied 5VDC and measured it at points on the board.

General Functionality

The first function of the board we tested was the ICSP programming capability. After a few minor issues with device signatures, we were able to burn the bootloader onto each PCB. After the bootloader was burned onto the board we uploaded our firmware and tested the Bluetooth connectivity. We were able to connect to the board with no issues, so then we moved onto testing all of the additional functionality.

We wanted to make sure the microprocessor was able to output a clean PWM signal. In the figure below you can see the PWM signal generated by the microcontroller

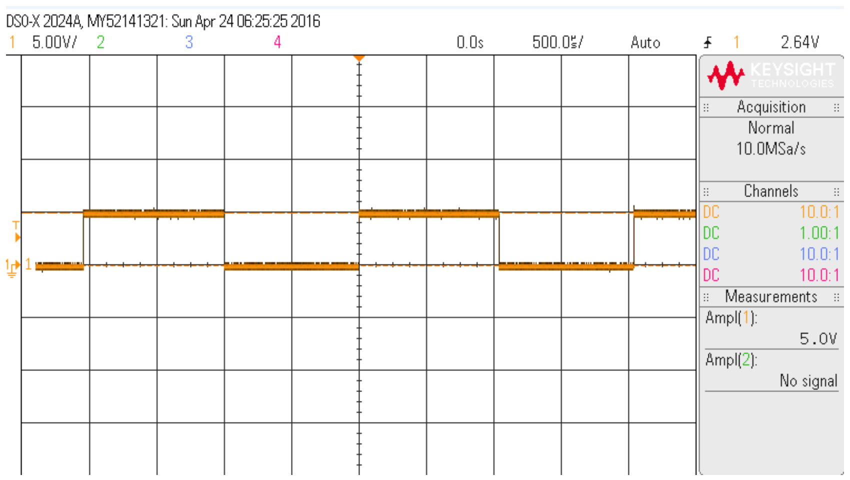


Figure 19. *Pulse Width Modulation Signal Used for Light Dimming*

After testing the PWM output, we made sure that the hardware switches and dimmers operated properly. During testing, we experienced issues with debouncing the hardware switches and dimmers. This caused us to adjust the delays before reading data into the microcontroller from the hardware.

Bluetooth Signal Range Testing

In order to test the effective range of the Bluetooth module, we simply tested whether or not we could connect to the microcontroller via Bluetooth at a certain distance. The maximum range of the Bluetooth module is 10 meters.

Power consumption

The last testing we performed is measuring the power consumption during different states that the microcontroller. The microcontroller draws around 15mA of current with a 5V supply with nothing connected giving a quiescent power consumption of 75mW. The When the PWM output is at full duty cycle, it draws around 30.1mA of current at 5V giving a power consumption of 150.5mW.

Mobile Application

iOS

Testing mobile applications is an arduous task, especially for those who are not yet accustomed with the system. The methods used to test the iOS app were somewhat naive in nature, as these included printing the different states of the app as it transitioned between establishing a connection, forming a command text, and sending it over to the microcontroller.

Testing became easier when the user's input became restricted to a smaller number of possibilities, i.e. brightness and whether the lights were on or off.
Most tests were performed under a simulated environment, where the test equipment included a simple breadboard with an Arduino Uno and a Bluetooth chip. The code running on the microcontroller was equivalent to the final microcontroller code that would be featured in our prototype. Therefore, any successful tests that ran on the test microcontroller were deemed as valid for the final product.

Android

Testing for Android systems was done mostly by trial-and-error. Once the user’s input was restricted by the GUI, there was a finite set of discrete inputs. While not all 1020 distinct cases were explicitly tested, several combinations of on/off, address, and brightness were selected, and it was verified that the bulb exhibited the expected behavior. In this manner, only usability tests were performed.

Rather than the final product, the test equipment was a simple breadboard with an Arduino Uno and a Bluetooth chip. The microcontroller code that the Uno was running was equivalent to the final microcontroller code that would be featured in our prototype. Once it was confirmed that the Android application would communicate with the prototype as well as it communicated with the testbed, it was evident that the tests were still valid.

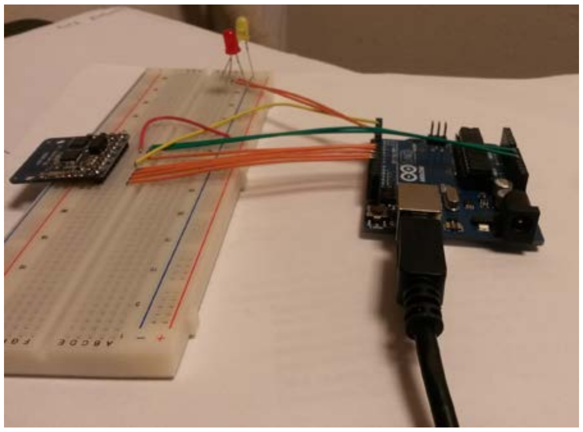


Figure 20. *Setup Used to Test Android Communication via BLE.*

LED Bulb

In order to test the correct operation of the LED bulb, we utilized a spectrometer to measure the spectrum of the Bulb. We were then able to measure the spectrum of a store bought LED 60W bulb and compare the two. This comparison is shown in ***Figure 23.*** The intensity of each bulb is normalized to their peak values in order to accurately compare the two. The device utilized to measure the spectrum is shown in ***Figure 24.***

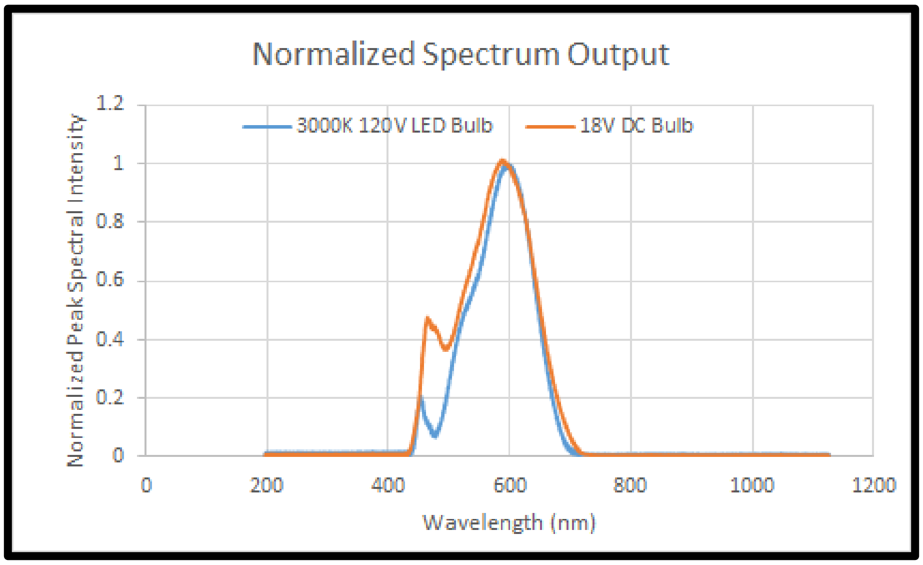


Figure 23. *Normalized Light Spectrum*

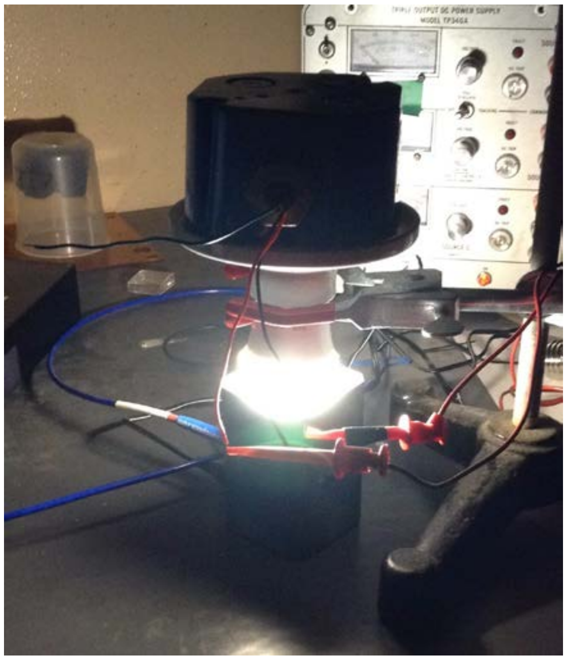


Figure 24. *Test Setup Used to Measure Light Spectrum.*

Appendix I: Operation Manual

To properly use the lights on the software side, first install either the Android or iOS applications. The applications mimic each other in functionality and appearance; please disregard any differences between images displayed here and what you see on your screen.
Launch the application you installed.
Select “Connect”. If you are prompted with a pop-up, select GT_SD.
Select Lighting.

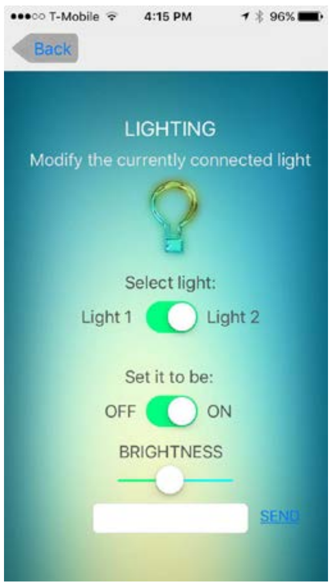
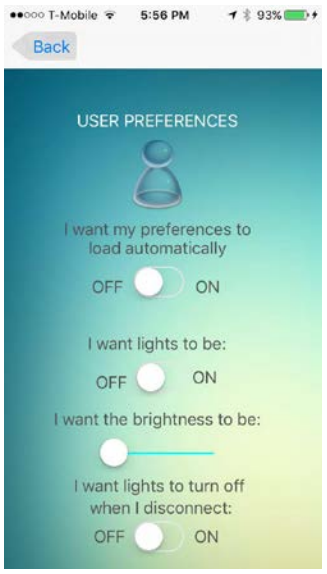


Figure 25.

Use the widgets (on/off switch and brightness slider) to adjust lighting to taste. If you desire more advanced options, you may also open the preferences screen.



USER PREFERENCES

I want my preferences to load automatically

OFF ON

I want lights to be:

OFF ON

I want the brightness to be:

I want lights to turn off when I disconnect:

OFF ON

Figure 26.

This screen allows you to control the lights' state when you connect/disconnect from the microcontroller.

When finished, disconnect or close the app.

A user also has the ability to control the lights using hardware switches and dimmers that are mounted in the wall. A user can adjust the brightness of the light bulb by turning the dimmer knob. The user can turn on and off a light bulb by switching the on/off switch mounted in the wall. The hardware switches will take precedent over the mobile applications commands. Additionally, there is a hardware reset pushbutton on the board that is used to reset the hardware given any hangups with the code or connectivity issues.

Embedded Systems Maintenance

In order to update the software on the board, there is a header used for ICSP programming. An Atmel programmer can be used to flash a bootloader or any microcontroller code necessary. This allows for easy software updates to the system if needed.

Appendix II: Alternative Design Versions

Mobile Application

There were a couple of features that we would have liked to have added, but didn't have enough time to do so. Currently, a connection is established with the closest proximity Bluetooth microcontroller, ideally however, the app would be running in the background and would automatically connect with a microcontroller when the user entered a room. The idea seems straightforward enough, but is actually quite complex. How can you define what a room is and whether you are currently inside it or not? It is hard for the app to tell whether you are physically inside a room or whether you are simply close to it, but have not yet actually entered it. One possibility is to use iBeacons in every room and use their signals to triangulate a user's position. Depending on how close the user is, establishing a connection with the microcontroller.

The second feature was to supply the user with power statistics, such as how much energy/money they are saving by using our lightbulbs and how much power has been released in a given time span. For that, we'd have to poll information from the microcontroller and use the results to generate and display statistics.

Microcontroller/Bluetooth

The microcontroller board had two prototypes throughout the course of the design process. The first prototype was just a simple breadboard implementation that only featured the Arduino Uno and the Adafruit Bluetooth breakout board. This prototype was used mainly to test the Bluetooth interface with the app. We were able to develop some rudimentary code that had this functionality. The second prototype had a similar setup, but a DS1307 real time clock breakout board was added as well as some hardware switches and dimmers. Testing and verifying the functionality of this system allowed us to implement the final design on our printed circuit boards.

Power Supply

Due to time restrictions, a fully offline power supply was not able to be developed and soldered, although the design was completed. An interim power supply consisting of a small 60 Hz transformer was used and placed inside the wall to step the voltage down from 120V AC to 13.1V AC. After stepping down the voltage, the supply was rectified and a buck converter was used to provide and regulate for the proper voltage. A linear post regulator was then used to provide the correct voltage for the microcontroller and Bluetooth. If there were sufficient time left, an isolated flyback converter would be implemented to increase efficiency and shrink the area required for the circuit.

Appendix III: Other Considerations

Part of the effort to make a working project was in fabricating the LED bulb fabrication. The first part of fabricating the PCB was to print the layout onto a clear projector screen (***Figure 27*).** Once the pattern is printed, it can then be taped down to a PCB board that has photoresist applied. After exposing the photoresist, the photoresist and copper can then be etched away (***Figure 28****)*. Once the copper traces are on, it is a matter of cutting out the PCB boards (***Figure 29****)*. With the boards cut out, the LEDs can then be soldered onto the boards (***Figure 30****)* and assembled into the bulb case (***Figure 31****)*. The case of the bulbs were salvaged from existing LED bulbs. The finished bulb is shown in ***Figure 32****.*

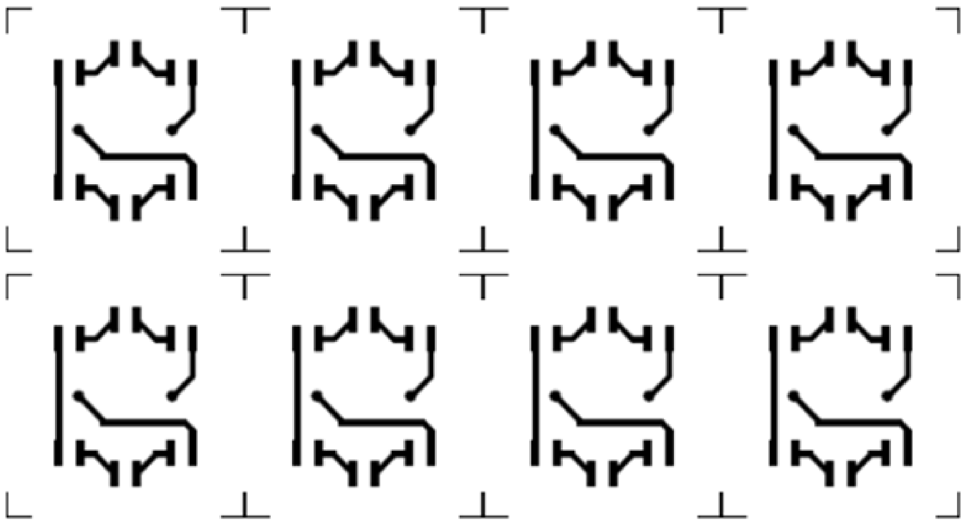


Figure 27. *PCB printout for fabrication.*

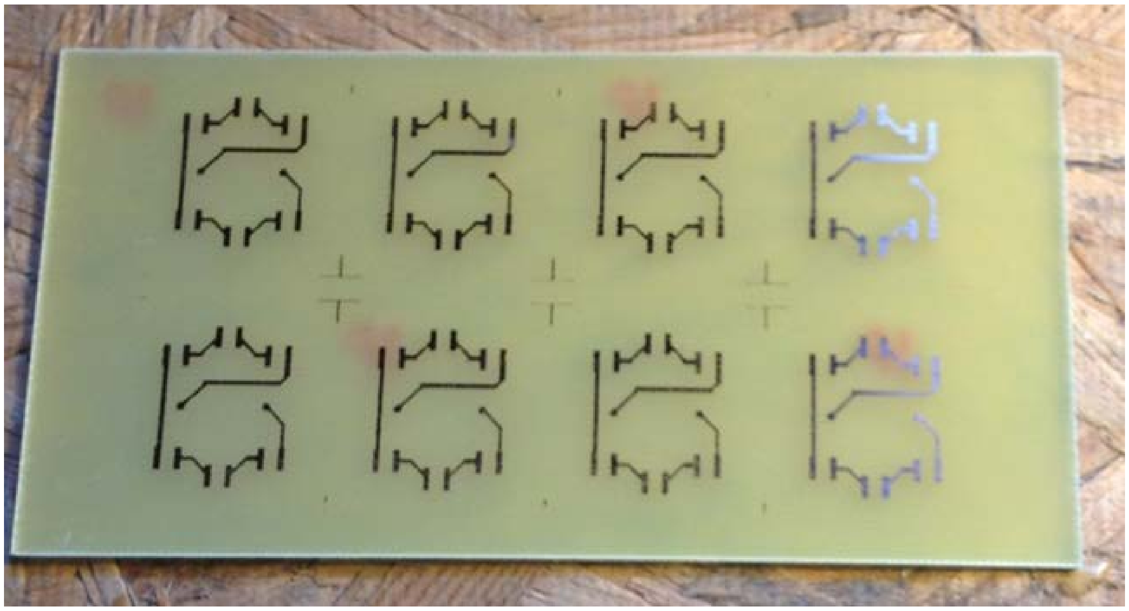


Figure 28. *PCB pattern in copper on board.*

Figure 29. *Cutting out* PCB *board.*

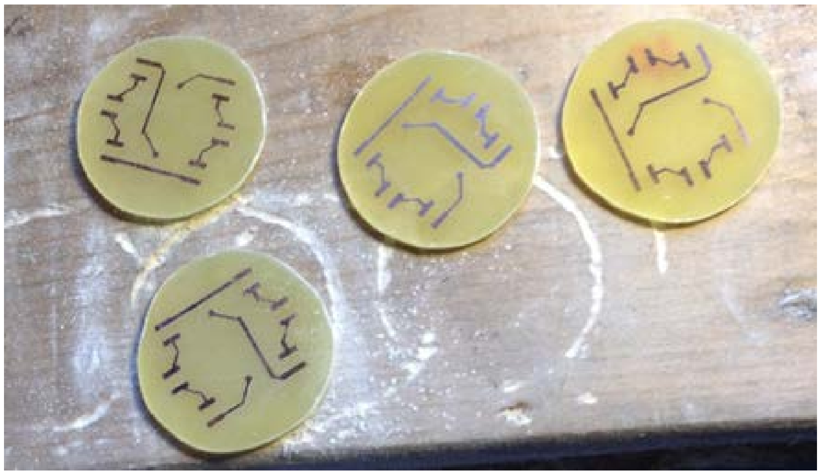


Figure 30. *Cut out* PCB *boards.*

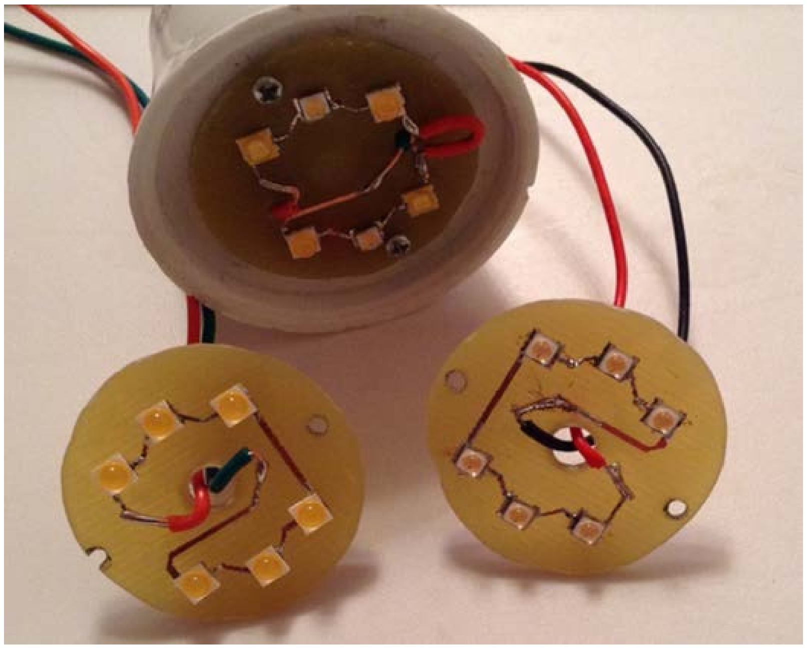


Figure 31. *Soldered* PCB *boards being assembled into light case.*

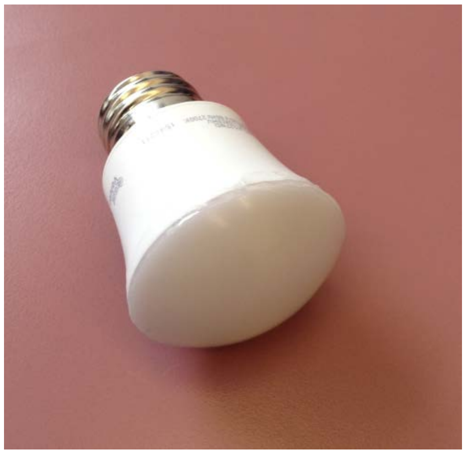
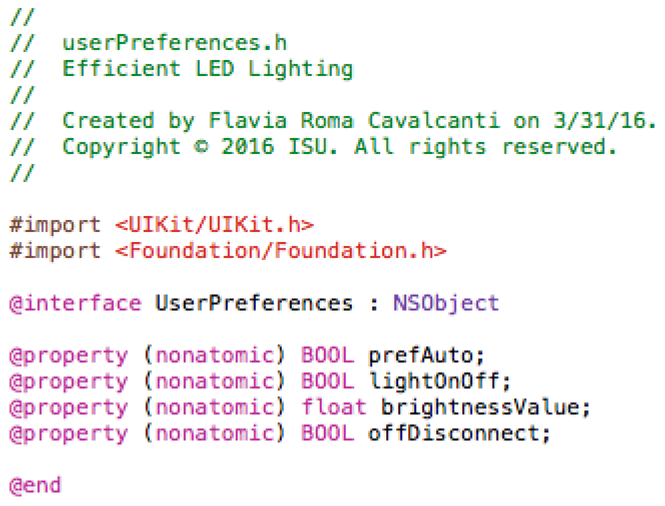


Figure 32. *Finished LED bulb.*

Appendix IV: Code

iOS Mobile Application

Model



```
//
//  userPreferences.h
//  Efficient LED Lighting
//
//  Created by Flavia Roma Cavalcanti on 3/31/16.
//  Copyright © 2016 ISU. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>

@interface UserPreferences : NSObject

@property (nonatomic) BOOL prefAuto;
@property (nonatomic) BOOL lightOnOff;
@property (nonatomic) float brightnessValue;
@property (nonatomic) BOOL offDisconnect;

@end
```

```
//
//  userPreferences.m
//  Efficient LED Lighting
//
//  Created by Flavia Roma Cavalcanti on 3/31/16.
//  Copyright © 2016 ISU. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "UserPreferences.h"

@implementation UserPreferences

#define PREF_SETTINGS_KEY @"Pref_Settings_Key"
#define PREF_STATE_KEY @"Pref_State_Key"
#define LIGHT_STATE_KEY @"Light_State_Key"
#define BRIGHTNESS_STATE_KEY @"Brightness_State_Key"
#define ON_OFF_DISC_KEY @"On_Off_Disc_Key"

- (BOOL)boolValueForKey:(NSString *)key withDefault:(BOOL)defaultValue
{
    NSDictionary *pref = [[NSUserDefaults standardUserDefaults] dictionaryForKey:
        PREF_SETTINGS_KEY];
    if (!pref) return defaultValue;
    if (![[pref allKeys] containsObject:key]) return defaultValue;
    return [pref[key] boolValue];
}

- (float)floatValueForKey:(NSString *)key withDefault:(float)defaultValue
{
    NSDictionary *pref = [[NSUserDefaults standardUserDefaults] dictionaryForKey:
        PREF_SETTINGS_KEY];
    if (!pref) return defaultValue;
    if (![[pref allKeys] containsObject:key]) return defaultValue;
    return [pref[key] floatValue];
}

- (BOOL)prefAuto
{
    return [self boolValueForKey:PREF_STATE_KEY withDefault:NO];
}

- (BOOL)lightOnOff
{
    return [self boolValueForKey:LIGHT_STATE_KEY withDefault:NO];
}

- (float) brightnessValue
{
    return [self floatValueForKey:BRIGHTNESS_STATE_KEY withDefault:0];
}
```

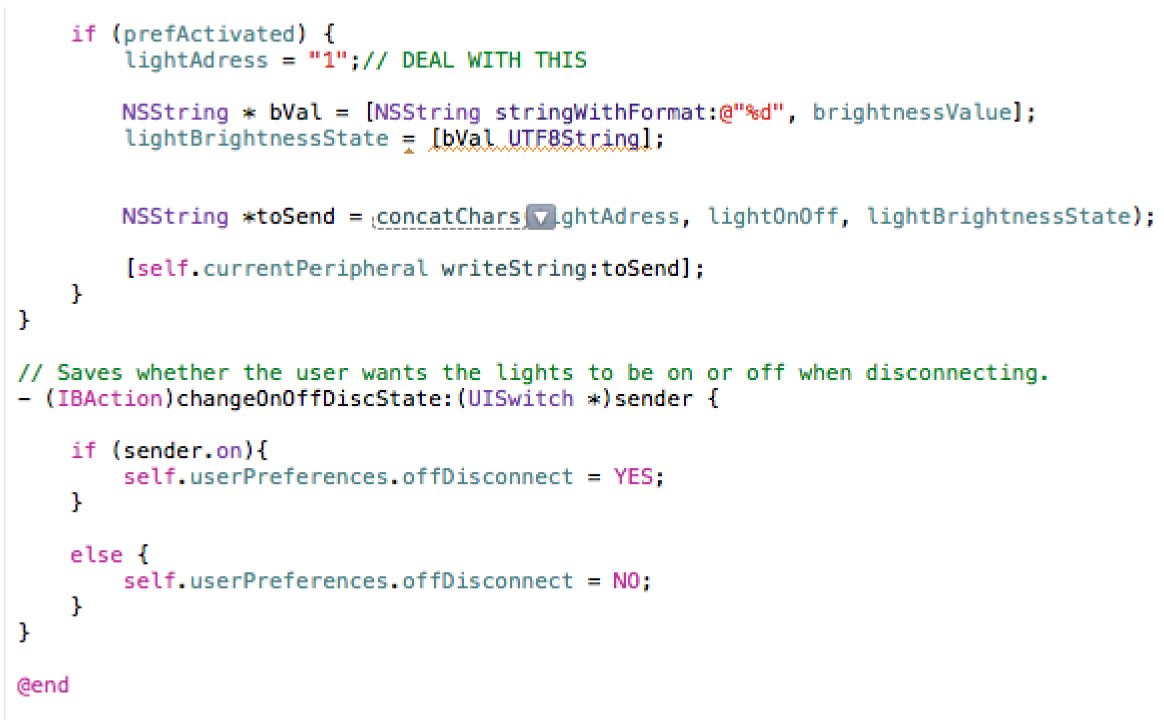
```
- (BOOL)prefAuto
{
    return [self boolValueForKey:PREF_STATE_KEY withDefault:NO];
}

- (BOOL)lightOnOff
{
    return [self boolValueForKey:LIGHT_STATE_KEY withDefault:NO];
}

- (float) brightnessValue
{
    return [self floatValueForKey:BRIGHTNESS_STATE_KEY withDefault:0];
}

- (BOOL) offDisconnect
{
    return [self boolValueForKey:ON_OFF_DISC_KEY withDefault:NO];
}

- (void)setBoolValue:(BOOL)value forKey:(NSString *)key
{
    NSMutableDictionary *pref = [[[NSUserDefaults standardUserDefaults] dictionaryForKey:
        PREF_SETTINGS_KEY] mutableCopy];
    if (!pref) {
        pref = [[NSMutableDictionary alloc] init];
    }
    pref[key] = @(value);
    [[NSUserDefaults standardUserDefaults] setObject:pref
                                              forKey:PREF_SETTINGS_KEY];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (void)setFloatValue:(float)value forKey:(NSString *)key
{
    NSMutableDictionary *pref = [[[NSUserDefaults standardUserDefaults] dictionaryForKey:
        PREF_SETTINGS_KEY] mutableCopy];
    if (!pref) {
        pref = [[NSMutableDictionary alloc] init];
    }
    pref[key] = @(value);
    [[NSUserDefaults standardUserDefaults] setObject:pref
                                              forKey:PREF_SETTINGS_KEY];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (void)setPrefAuto:(BOOL)prefAuto
{
    [self setBoolValue:prefAuto forKey:PREF_STATE_KEY];
}
```

```
- (void)setPrefAuto:(BOOL)prefAuto
{
    [self setBoolValue:prefAuto forKey:PREF_STATE_KEY];
}

- (void)setLightOnOff:(BOOL)lightOnOff
{
    [self setBoolValue:lightOnOff forKey:LIGHT_STATE_KEY];
}

- (void)setBrightnessValue:(float)brightnessValue{
    [self setFloatValue:brightnessValue forKey:BRIGHTNESS_STATE_KEY];
}

- (void)setOffDisconnect:(BOOL)offDisconnect{
    [self setBoolValue:offDisconnect forKey:ON_OFF_DISC_KEY];
}

@end
```

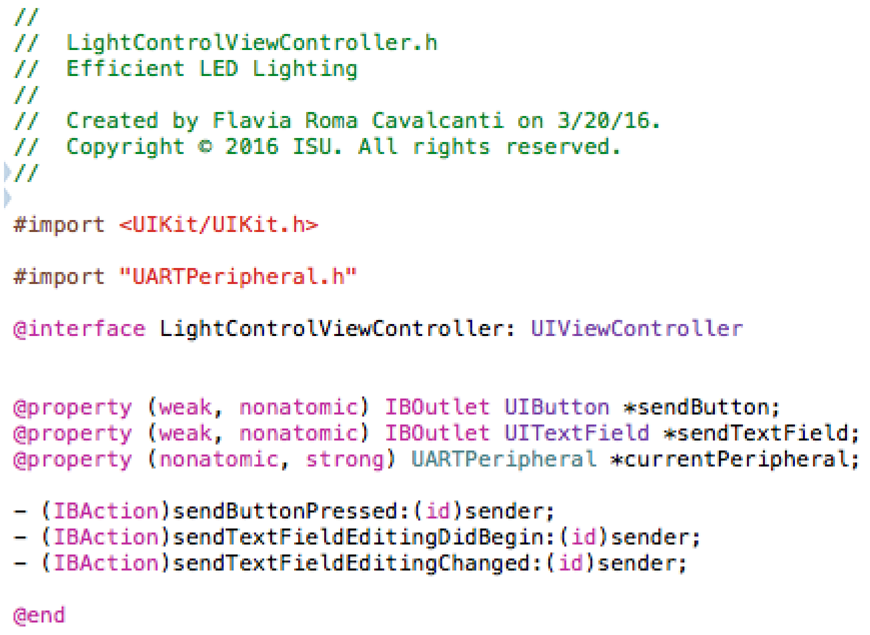
View Controller

```
//
//  UserPrefViewController.h
//  Efficient LED Lighting
//
//  Created by Flavia Roma Cavalcanti on 3/27/16.
//  Copyright © 2016 ISU. All rights reserved.
//

#import <UIKit/UIKit.h>

#import "UARTPeripheral.h"

@interface UserPrefViewController : UIViewController

@property (nonatomic, strong) UARTPeripheral *currentPeripheral;

@property (weak, nonatomic) IBOutlet UISwitch *prefState;

@property (weak, nonatomic) IBOutlet UISwitch *lightState;

@property (weak, nonatomic) IBOutlet UISlider *brightnessState;

@property (weak, nonatomic) IBOutlet UISwitch *onOffDisconnectState;


@end
```

```
//
//  UserPrefViewController.m
//  Efficient LED Lighting
//
//  Created by Flavia Roma Cavalcanti on 3/27/16.
//  Copyright © 2016 ISU. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "UserPrefViewController.h"
#import "UserPreferences.h"

@interface UserPrefViewController()

@property (strong, nonatomic) UserPreferences *userPreferences;

@end

@implementation UserPrefViewController

- (UserPreferences *) userPreferences {
    if (!_userPreferences) _userPreferences = [[UserPreferences alloc] init];
    return _userPreferences;
}

char * lightAdress = "1"; // Change later to be modified by a light picker
char * lightOnOff = "1"; // On/Off button
char * lightBrightnessState = "111"; // Light brightness
int brightnessValue = 0;
BOOL prefActivated = YES;


// Return a concatenation of the addr, onOff, and brightness chars
NSString *concatChars (char * addr, char * onOff, char * brightness){

    NSString * addrString = [[NSString alloc]initWithUTF8String:addr];
    NSString * onOffString = [[NSString alloc]initWithUTF8String:onOff];
    NSString * brightnessString = [[NSString alloc]initWithUTF8String:brightness];

    NSString * temp = [addrString stringByAppendingString:onOffString];
    NSString * ret = [temp stringByAppendingString:brightnessString];

    return ret;
}

// If the view loads, modify the view to match the last previously saved preference data
- (void)viewDidLoad{
    [super viewDidLoad];

    self.lightState.on = (self.userPreferences.lightOnOff);
    self.prefState.on = (self.userPreferences.prefAuto);
    self.brightnessState.value = self.userPreferences.brightnessValue;
    self.onOffDisconnectState.on = self.userPreferences.offDisconnect;
}
```



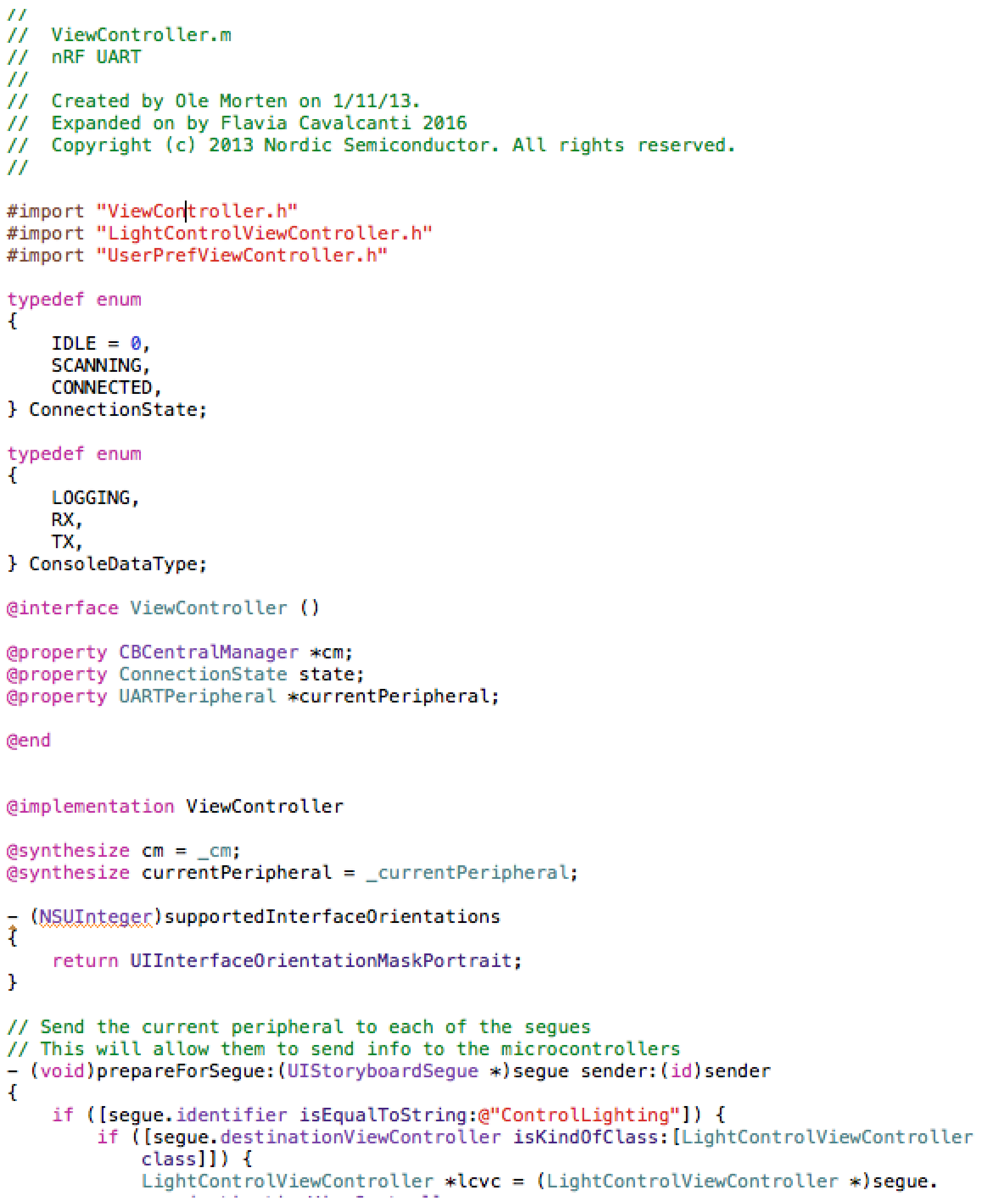
```
    if (prefActivated) {
        lightAdress = "1";// DEAL WITH THIS

        NSString * bVal = [NSString stringWithFormat:@"%d", brightnessValue];
        lightBrightnessState = [bVal UTF8String];


        NSString *toSend = concatChars(▼ghtAdress, lightOnOff, lightBrightnessState);

        [self.currentPeripheral writeString:toSend];
    }
}

// Saves whether the user wants the lights to be on or off when disconnecting.
- (IBAction)changeOnOffDiscState:(UISwitch *)sender {

    if (sender.on){
        self.userPreferences.offDisconnect = YES;
    }

    else {
        self.userPreferences.offDisconnect = NO;
    }
}

@end
```



```
//
//  LightControlViewController.h
//  Efficient LED Lighting
//
//  Created by Flavia Roma Cavalcanti on 3/20/16.
//  Copyright © 2016 ISU. All rights reserved.
//

#import <UIKit/UIKit.h>

#import "UARTPeripheral.h"

@interface LightControlViewController: UIViewController


@property (weak, nonatomic) IBOutlet UIButton *sendButton;
@property (weak, nonatomic) IBOutlet UITextField *sendTextField;
@property (nonatomic, strong) UARTPeripheral *currentPeripheral;

- (IBAction)sendButtonPressed:(id)sender;
- (IBAction)sendTextFieldEditingDidBegin:(id)sender;
- (IBAction)sendTextFieldEditingChanged:(id)sender;

@end
```

```
//
//  LightControlViewController.m
//  Efficient LED Lighting
//
//  Created by Flavia Roma Cavalcanti on 3/21/16.
//  Copyright © 2016 ISU. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "LightControlViewController.h"

char * addr = "1"; //Will be modified by the light picker
char * onOff = "1"; //Will be modified by the on/off button
char * brightness = "111"; //Will be modified by the slider
int brightnessVal = 0;
BOOL debug = YES; //if set to true, will print out values of variables

@interface LightControlViewController ()
@end

@implementation LightControlViewController
@synthesize currentPeripheral = _currentPeripheral;


//Concatenate three char pointers into a single NSString object
NSString* concatenateChar (char * addr, char * onOff, char * brightness){
    NSString * addrString = [[NSString alloc]initWithUTF8String:addr];
    NSString * onOffString = [[NSString alloc]initWithUTF8String:onOff];
    NSString * brightnessString = [[NSString alloc]initWithUTF8String:brightness];

    if (debug){
        NSLog(@"brightness");
        NSLog(brightnessString);
    }

    NSString * temp = [addrString stringByAppendingString:onOffString];
    NSString * ret = [temp stringByAppendingString:brightnessString];

    if (debug){
        NSLog(@"ret");
        NSLog(ret);
    }
    return ret;
}

// Slider for the brightness -- modifies the dimness of the connected bulb
- (IBAction)BrightnessSlider:(UISlider *)sender {
    float valueF = sender.value;
    int roundedFloat = lroundf(valueF);
    brightnessVal = roundedFloat;

    NSString * val = [NSString stringWithFormat:@"%d", roundedFloat];
    brightness = [val UTF8String];
    NSString *toSend = concatenateChar(addr, onOff, brightness);//[appendStrings str1:addr
        str2:onOff, str3:brightness];

    if (debug){
        NSLog(@"Test");
        NSLog(toSend);
        NSLog(val);
    }

    self.sendTextField.text = toSend;
    [self.currentPeripheral writeString:toSend];
}

// Modifies which lightbulb we're currently modifying
// The microncontroller currently supports up to two lightbulbs
- (IBAction)lightAddressSwitch:(UISwitch *)sender {

    if (sender.on){
        addr = "1";
    }

    else {
        addr = "0";
    }

    NSString * bVal = [NSString stringWithFormat:@"%d", brightnessVal];
    brightness = [bVal UTF8String];

    NSString *toSend = concatenateChar(addr, onOff, brightness);//[appendStrings str1:addr
        str2:onOff, str3:brightness];

    self.sendTextField.text = toSend;
    [self.currentPeripheral writeString:toSend];
}

// Whether the currently connected light is on or off
- (IBAction)OnOffSwitch:(UISwitch *)sender {
    if (debug){
        NSLog(@"before");
        NSLog([NSString stringWithFormat:@"%c", brightness]);
    }

    if(sender.on){
        onOff = "1";
    }
    else{
        onOff = "0";
    }

    NSString * bVal = [NSString stringWithFormat:@"%d", brightnessVal];
    brightness = [bVal UTF8String];
    if (debug){
        NSLog(@"after");
        NSLog([NSString stringWithFormat:@"%c", brightness]);
    }

    NSString *toSend = concatenateChar(addr, onOff, brightness);//[appendStrings str1:addr
        str2:onOff, str3:brightness];
    if (debug){
        NSLog(toSend);
    }
    self.sendTextField.text = toSend;
    [self.currentPeripheral writeString:toSend];
}

- (IBAction)sendTextFieldEditingDidBegin:(id)sender {
    //    [self.tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForRow:0 inSection:
        2] atScrollPosition:UITableViewScrollPositionBottom animated:YES];
    //    [self.tableView setContentOffset:CGPointMake(0, 220) animated:YES];
}

- (IBAction)sendTextFieldEditingChanged:(id)sender {
    if (self.sendTextField.text.length > 20)
    {
        [self.sendTextField setBackgroundColor:[UIColor redColor]];
    }
    else
    {
        [self.sendTextField setBackgroundColor:[UIColor whiteColor]];
    }
}

- (BOOL) textFieldShouldReturn:(UITextField *)textField
{
    [self sendButtonPressed:textField];
    return YES;
}
- (IBAction)sendButtonPressed:(id)sender {
    [self.sendTextField resignFirstResponder];

    if (self.sendTextField.text.length == 0)
    {
        return;
    }
    NSLog(@" %@", self.sendTextField.text);

    //[self addTextToConsole:self.sendTextField.text dataType:TX];

    [self.currentPeripheral writeString:self.sendTextField.text];
}


@end
```

```
//
//  ViewController.h
//  nRF UART
//
//  Created by Ole Morten on 1/11/13.
//  Expanded by Flavia Cavalcanti 2016
//  Copyright (c) 2013 Nordic Semiconductor. All rights reserved.
//

#import <UIKit/UIKit.h>

#import "UARTPeripheral.h"

@interface ViewController: UIViewController
//@property (strong, nonatomic) IBOutlet UITableView *tableView;
@property (weak, nonatomic) IBOutlet UIButton *connectButton;

- (IBAction)connectButtonPressed:(id)sender;

@property (nonatomic) BOOL prefActivated;
@property (nonatomic) float brightness;
@property (nonatomic) BOOL onOff;

@end
```



```
//
//  ViewController.m
//  nRF UART
//
//  Created by Ole Morten on 1/11/13.
//  Expanded on by Flavia Cavalcanti 2016
//  Copyright (c) 2013 Nordic Semiconductor. All rights reserved.
//

#import "ViewController.h"
#import "LightControlViewController.h"
#import "UserPrefViewController.h"

typedef enum
{
    IDLE = 0,
    SCANNING,
    CONNECTED,
} ConnectionState;

typedef enum
{
    LOGGING,
    RX,
    TX,
} ConsoleDataType;

@interface ViewController ()

@property CBCentralManager *cm;
@property ConnectionState state;
@property UARTPeripheral *currentPeripheral;

@end


@implementation ViewController

@synthesize cm = _cm;
@synthesize currentPeripheral = _currentPeripheral;

- (NSUInteger)supportedInterfaceOrientations
{
    return UIInterfaceOrientationMaskPortrait;
}

// Send the current peripheral to each of the segues
// This will allow them to send info to the microcontrollers
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    if ([segue.identifier isEqualToString:@"ControlLighting"]) {
        if ([segue.destinationViewController isKindOfClass:[LightControlViewController
            class]]) {
            LightControlViewController *lcvc = (LightControlViewController *)segue.
            LightControlViewController *lcvc = (LightControlViewController *)segue.
                destinationViewController;
            lcvc.currentPeripheral = self.currentPeripheral;
        }
    }

    if ([segue.identifier isEqualToString:@"UserPrefSegue"]){
        if ([segue.destinationViewController isKindOfClass:[UserPrefViewController class]])
            {
            UserPrefViewController *upvc = (UserPrefViewController *)segue.
                destinationViewController;
            upvc.currentPeripheral = self.currentPeripheral;
        }
    }
}


- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    self.cm = [[CBCentralManager alloc] initWithDelegate:self queue:nil];

    [self addTextToConsole:@"Started application" dataType:LOGGING];

}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

- (IBAction)connectButtonPressed:(id)sender
{
    //[self.sendTextField resignFirstResponder];

    switch (self.state) {
        case IDLE:
            self.state = SCANNING;

            NSLog(@"Started scan ...");
            [self.connectButton setTitle:@"SCANNING ..." forState:UIControlStateNormal];

            [self.cm scanForPeripheralsWithServices:@[UARTPeripheral.uartServiceUUID]
                options:@{CBCentralManagerScanOptionAllowDuplicatesKey: [NSNumber
                numberWithBool:NO]}];
            break;

        case SCANNING:
            self.state = IDLE;

            NSLog(@"Stopped scan");
            [self.connectButton setTitle:@"CONNECT" forState:UIControlStateNormal];
```

```
- (void) didReadHardwareRevisionString:(NSString *)string
{
    [self addTextToConsole:[NSString stringWithFormat:@"Hardware revision: %@", string]
        dataType:LOGGING];
}

- (void) didReceiveData:(NSString *)string
{
    [self addTextToConsole:string dataType:RX];
}

- (void) addTextToConsole:(NSString *) string dataType:(ConsoleDataType) dataType
{
    NSString *direction;
    switch (dataType)
    {
        case RX:
            direction = @"RX";
            break;

        case TX:
            direction = @"TX";
            break;

        case LOGGING:
            direction = @"Log";
    }

    NSDateFormatter *formatter;
    formatter = [[NSDateFormatter alloc] init];
    [formatter setDateFormat:@"HH:mm:ss.SSS"];

}

- (void) centralManagerDidUpdateState:(CBCentralManager *)central
{
    if (central.state == CBCentralManagerStatePoweredOn)
    {
        [self.connectButton setEnabled:YES];
    }

}

- (void) centralManager:(CBCentralManager *)central didDiscoverPeripheral:(CBPeripheral *)
    peripheral advertisementData:(NSDictionary *)advertisementData RSSI:(NSNumber *)RSSI
{
    NSLog(@"Did discover peripheral %@", peripheral.name);
    [self.cm stopScan];

    self.currentPeripheral = [[UARTPeripheral alloc] initWithPeripheral:peripheral
        delegate:self];
    self.currentPeripheral = [[UARTPeripheral alloc] initWithPeripheral:peripheral
        delegate:self];

    [self.cm connectPeripheral:peripheral options:
        @{CBConnectPeripheralOptionNotifyOnDisconnectionKey: [NSNumber numberWithBool:YES]}
        ];
}

- (void) centralManager:(CBCentralManager *)central didConnectPeripheral:(CBPeripheral *)
    peripheral
{
    NSLog(@"Did connect peripheral %@", peripheral.name);

    [self addTextToConsole:[NSString stringWithFormat:@"Did connect to %@", peripheral.name
        ] dataType:LOGGING];

    self.state = CONNECTED;
    [self.connectButton setTitle:@"DISCONNECT" forState:UIControlStateNormal];

    //[self.sendButton setUserInteractionEnabled:YES];
    //[self.sendTextField setUserInteractionEnabled:YES];

    if ([self.currentPeripheral.peripheral isEqual:peripheral])
    {
        [self.currentPeripheral didConnect];
    }

}

- (void) centralManager:(CBCentralManager *)central didDisconnectPeripheral:(CBPeripheral *
    )peripheral error:(NSError *)error
{
    NSLog(@"Did disconnect peripheral %@", peripheral.name);

    [self addTextToConsole:[NSString stringWithFormat:@"Did disconnect from %@, error code
        %ld", peripheral.name, (long)error.code] dataType:LOGGING];

    self.state = IDLE;
    [self.connectButton setTitle:@"Connect" forState:UIControlStateNormal];
    //[self.sendButton setUserInteractionEnabled:NO];
    //[self.sendTextField setUserInteractionEnabled:NO];

    if ([self.currentPeripheral.peripheral isEqual:peripheral])
    {
        [self.currentPeripheral didDisconnect];
    }
}


@end
```

UART System

```
//
//  AppDelegate.h
//  Efficient LED Lighting
//
//  Created by Flavia Roma Cavalcanti on 1/12/16.
//  Copyright © 2016 ISU. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;


@end
```

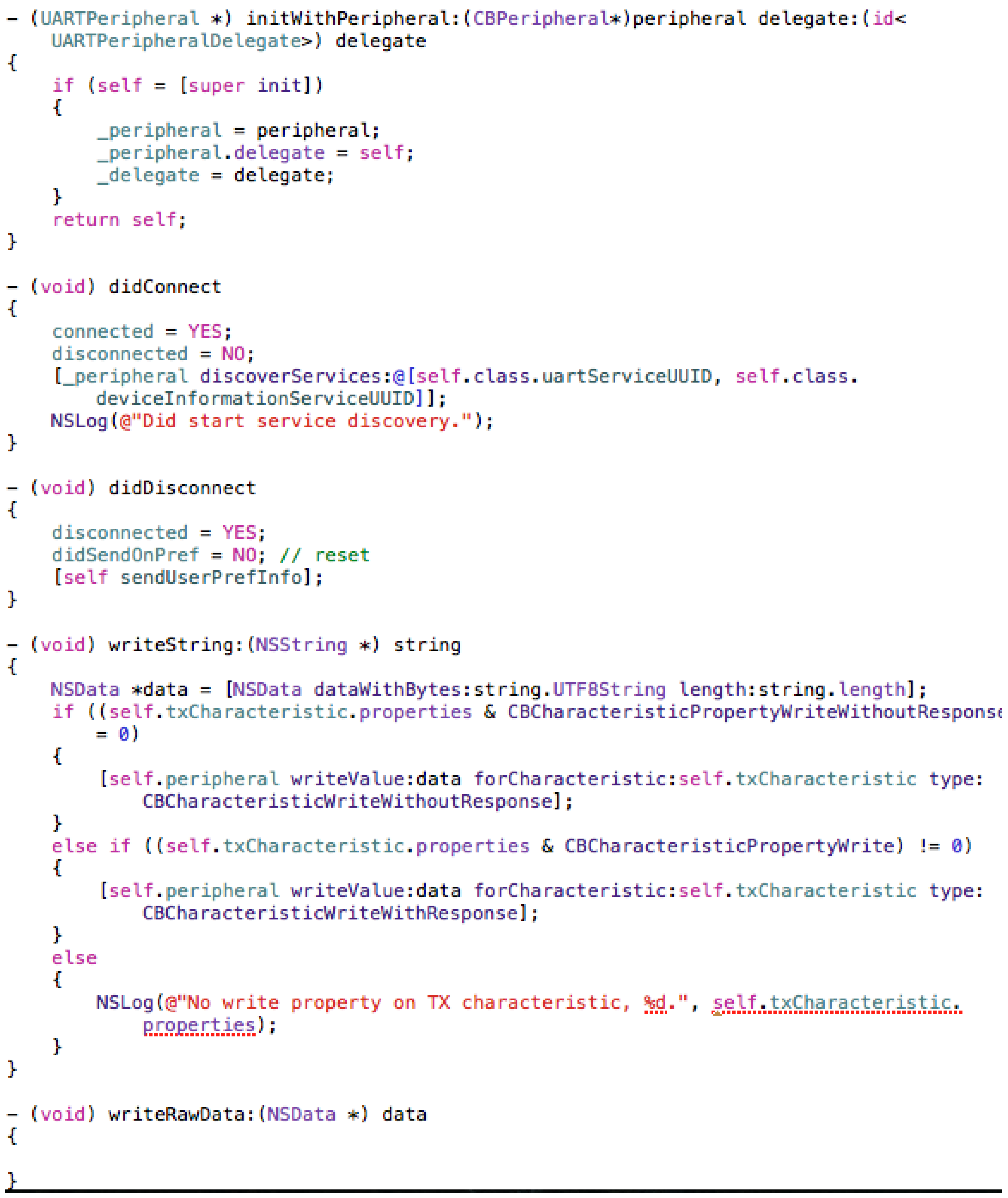
```
//
// Available for free download from : https://www.nordicsemi.com/eng/Products/Nordic-mobile
    -Apps/nRF-UART-App
//  Created by Ole Morten on 1/12/13.
//  Copyright (c) 2013 Nordic Semiconductor. All rights reserved.
//
//  AppDelegate.m
//  Efficient LED Lighting
//
//

#import "AppDelegate.h"

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
    (NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can
        occur for certain types of temporary interruptions (such as an incoming phone call
        or SMS message) or when the user quits the application and it begins the transition
        to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES
        frame rates. Games should use this method to pause the game.
}

- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate timers, and
        store enough application state information to restore your application to its
        current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of
        applicationWillTerminate: when the user quits.
}

- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the inactive state; here you
        can undo many of the changes made on entering the background.
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the application was
        inactive. If the application was previously in the background, optionally refresh
        the user interface.
}
```

```
- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the inactive state; here you
        can undo many of the changes made on entering the background.
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the application was
        inactive. If the application was previously in the background, optionally refresh
        the user interface.
}

- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save data if appropriate. See
        also applicationDidEnterBackground:.
}

@end
```

```
//
// Available for free download from : https://www.nordicsemi.com/eng/Products/Nordic-mobile
    -Apps/nRF-UART-App
//
//  UARTPeripheral.h
//  nRF UART
//
//  Created by Ole Morten on 1/12/13.
//  Copyright (c) 2013 Nordic Semiconductor. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "UserPreferences.h"
@import CoreBluetooth;

@protocol UARTPeripheralDelegate
- (void) didReceiveData:(NSString *) string;
@optional
- (void) didReadHardwareRevisionString:(NSString *) string;
@end


@interface UARTPeripheral : NSObject <CBPeripheralDelegate>

@property (nonatomic, strong)UserPreferences *userPreferences;

@property CBPeripheral *peripheral;
@property id<UARTPeripheralDelegate> delegate;

+ (CBUUID *) uartServiceUUID;

- (UARTPeripheral *) initWithPeripheral:(CBPeripheral*)peripheral delegate:(id<
    UARTPeripheralDelegate>) delegate;

- (void) writeString:(NSString *) string;

- (void) didConnect;
- (void) didDisconnect;
@end
```

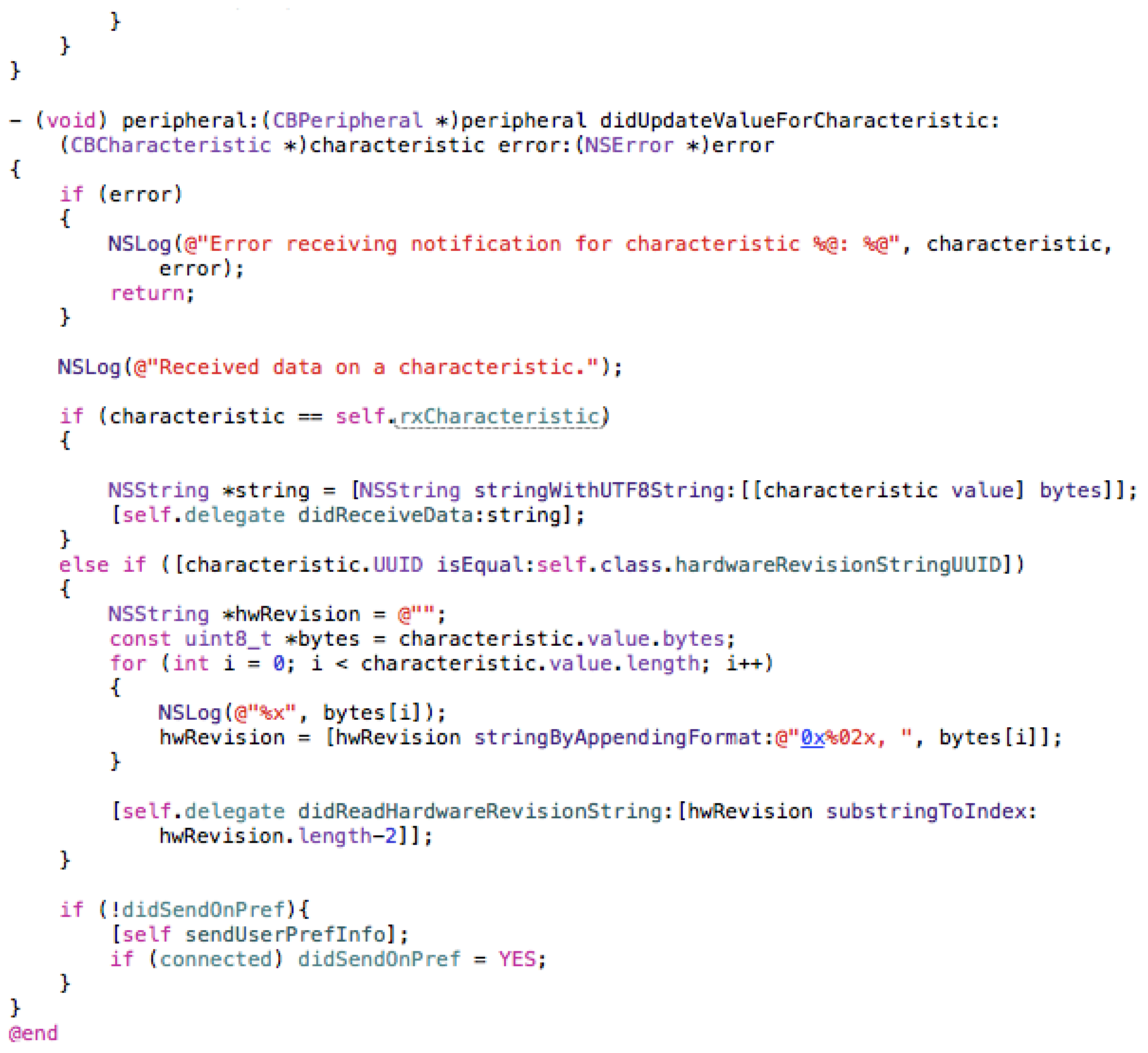
```
//
// Available for free download from : https://www.nordicsemi.com/eng/Products/Nordic-mobile
    -Apps/nRF-UART-App
//
//  UARTPeripheral.m
//  nRF UART
//
//  Created by Ole Morten on 1/12/13.
//  Copyright (c) 2013 Nordic Semiconductor. All rights reserved.
//

#import "UARTPeripheral.h"

@interface UARTPeripheral ()
@property CBService *uartService;
@property CBCharacteristic *rxCharacteristic;
@property CBCharacteristic *txCharacteristic;
@end

@implementation UARTPeripheral
@synthesize peripheral = _peripheral;
@synthesize delegate = _delegate;

@synthesize uartService = _uartService;
@synthesize rxCharacteristic = _rxCharacteristic;
@synthesize txCharacteristic = _txCharacteristic;

// I'm sure there is a better way to deal with this... heck
BOOL didSendOnPref = NO;
BOOL didSendOffPref = NO;
BOOL connected = NO;
BOOL disconnected = NO;

- (UserPreferences *) userPreferences {
    if (!_userPreferences) _userPreferences = [[UserPreferences alloc] init];
    return _userPreferences;
}

// Receives 3 char arrays and concatenates them into a single NSString
NSString *concatenateCharacters (char * addr, char * onOff, char * brightness){

    NSString * addrString = [[NSString alloc]initWithUTF8String:addr];
    NSString * onOffString = [[NSString alloc]initWithUTF8String:onOff];
    NSString * brightnessString = [[NSString alloc]initWithUTF8String:brightness];

    NSString * temp = [addrString stringByAppendingString:onOffString];
    NSString * ret = [temp stringByAppendingString:brightnessString];

    return ret;
}
```

```
// Fetches the user's saved preferences and sends that information to the currently
    connected light bulb
- (void) sendUserPrefInfo {
    char *lightsOn;
    char *lightBrightnessState;

    lightsOn = self.userPreferences.lightOnOff == YES ? "1" : "0";

    float valueF = lroundf(self.userPreferences.brightnessValue);
    NSString * val = [NSString stringWithFormat:@"%ld", lroundf(valueF)];
    lightBrightnessState = [val UTF8String];

    if (disconnected && self.userPreferences.offDisconnect){
        NSString *toSend = concatenateCharacters("1", "0", "255");
        NSLog(toSend);
        [self writeString:toSend];
    }

    else if (connected){
        NSString *toSend = concatenateCharacters("1", lightsOn, lightBrightnessState);
        NSLog(toSend);
        [self writeString:toSend];
    }

    NSLog(@"SENT");
}

+ (CBUUID *) uartServiceUUID
{
    return [CBUUID UUIDWithString:@"6e400001-b5a3-f393-e0a9-e50e24dcca9e"];
}

+ (CBUUID *) txCharacteristicUUID
{
    return [CBUUID UUIDWithString:@"6e400002-b5a3-f393-e0a9-e50e24dcca9e"];
}

+ (CBUUID *) rxCharacteristicUUID
{
    return [CBUUID UUIDWithString:@"6e400003-b5a3-f393-e0a9-e50e24dcca9e"];
}

+ (CBUUID *) deviceInformationServiceUUID
{
    return [CBUUID UUIDWithString:@"180A"];
}

+ (CBUUID *) hardwareRevisionStringUUID
{
    return [CBUUID UUIDWithString:@"2A27"];
}
```



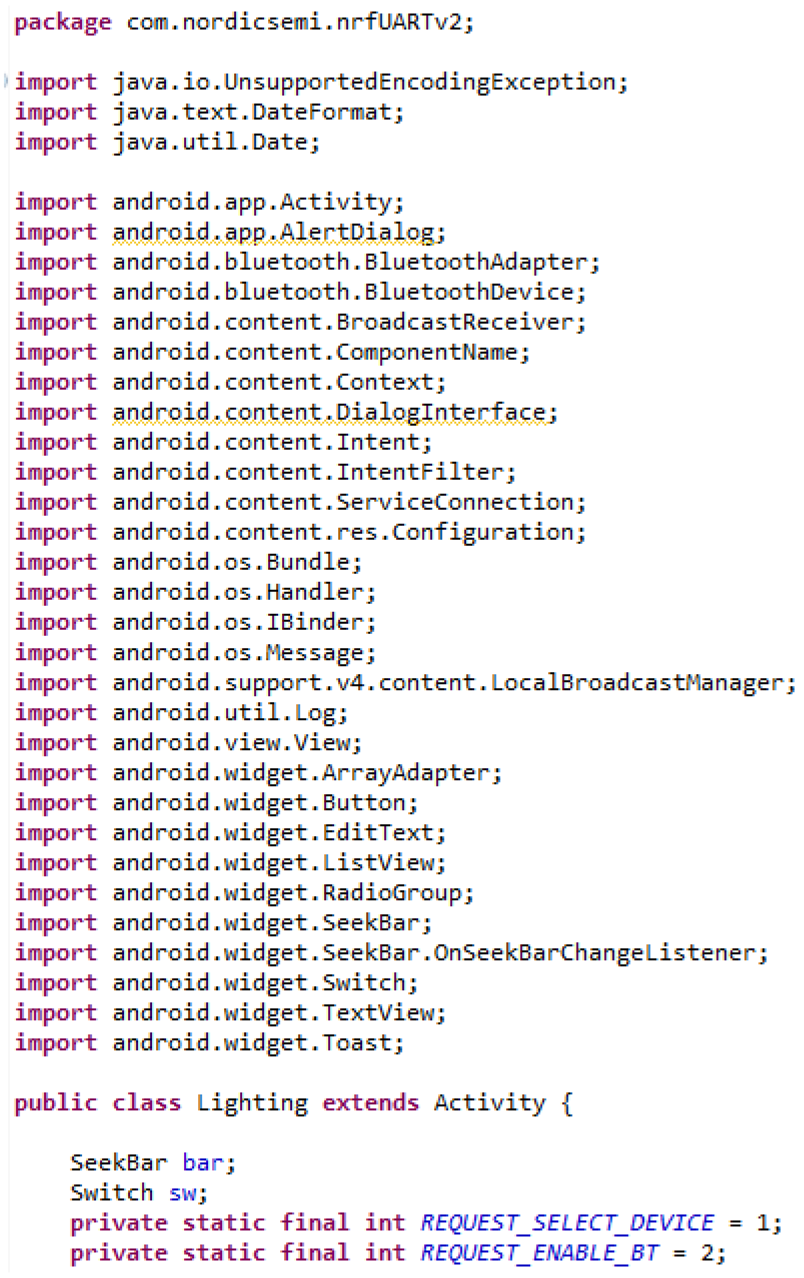
```
- (UARTPeripheral *) initWithPeripheral:(CBPeripheral*)peripheral delegate:(id<
    UARTPeripheralDelegate>) delegate
{
    if (self = [super init])
    {
        _peripheral = peripheral;
        _peripheral.delegate = self;
        _delegate = delegate;
    }
    return self;
}

- (void) didConnect
{
    connected = YES;
    disconnected = NO;
    [_peripheral discoverServices:@[self.class.uartServiceUUID, self.class.
        deviceInformationServiceUUID]];
    NSLog(@"Did start service discovery.");
}

- (void) didDisconnect
{
    disconnected = YES;
    didSendOnPref = NO; // reset
    [self sendUserPrefInfo];
}

- (void) writeString:(NSString *) string
{
    NSData *data = [NSData dataWithBytes:string.UTF8String length:string.length];
    if ((self.txCharacteristic.properties & CBCharacteristicPropertyWriteWithoutRespons
        = 0)
    {
        [self.peripheral writeValue:data forCharacteristic:self.txCharacteristic type:
            CBCharacteristicWriteWithoutResponse];
    }
    else if ((self.txCharacteristic.properties & CBCharacteristicPropertyWrite) != 0)
    {
        [self.peripheral writeValue:data forCharacteristic:self.txCharacteristic type:
            CBCharacteristicWriteWithResponse];
    }
    else
    {
        NSLog(@"No write property on TX characteristic, %d.", self.txCharacteristic.
            properties);
    }
}

- (void) writeRawData:(NSData *) data
{

}
```

```
- (void) peripheral:(CBPeripheral *)peripheral didDiscoverServices:(NSError *)error
{
    if (error)
    {
        NSLog(@"Error discovering services: %@", error);
        return;
    }

    for (CBService *s in [peripheral services])
    {
        if ([s.UUID isEqual:self.class.uartServiceUUID])
        {
            NSLog(@"Found correct service");
            self.uartService = s;

            [self.peripheral discoverCharacteristics:@[self.class.txCharacteristicUUID,
                self.class.rxCharacteristicUUID] forService:self.uartService];
        }
        else if ([s.UUID isEqual:self.class.deviceInformationServiceUUID])
        {
            [self.peripheral discoverCharacteristics:@[self.class.
                hardwareRevisionStringUUID] forService:s];
        }
    }
}

- (void) peripheral:(CBPeripheral *)peripheral didDiscoverCharacteristicsForService:
    (CBService *)service error:(NSError *)error
{
    if (error)
    {
        NSLog(@"Error discovering characteristics: %@", error);
        return;
    }

    for (CBCharacteristic *c in [service characteristics])
    {
        if ([c.UUID isEqual:self.class.rxCharacteristicUUID])
        {
            NSLog(@"Found RX characteristic");
            self.rxCharacteristic = c;

            [self.peripheral setNotifyValue:YES forCharacteristic:self.rxCharacteristic];
        }
        else if ([c.UUID isEqual:self.class.txCharacteristicUUID])
        {
            NSLog(@"Found TX characteristic");
            self.txCharacteristic = c;
        }
        else if ([c.UUID isEqual:self.class.hardwareRevisionStringUUID])
        {
            NSLog(@"Found Hardware Revision String characteristic");
            [self.peripheral readValueForCharacteristic:c];
```

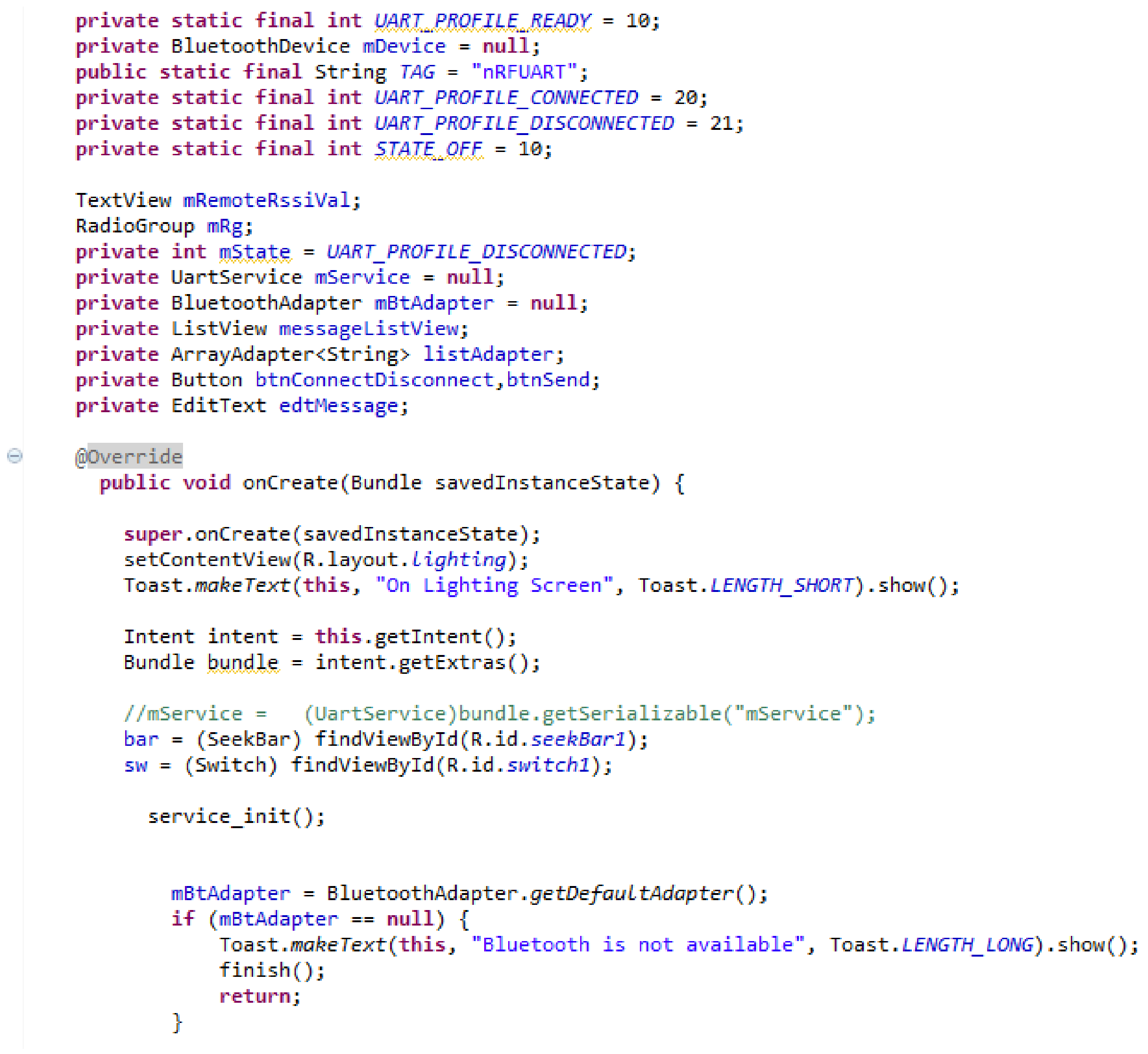


```
        }
    }
}

- (void) peripheral:(CBPeripheral *)peripheral didUpdateValueForCharacteristic:
    (CBCharacteristic *)characteristic error:(NSError *)error
{
    if (error)
    {
        NSLog(@"Error receiving notification for characteristic %@: %@", characteristic,
            error);
        return;
    }

    NSLog(@"Received data on a characteristic.");

    if (characteristic == self.rxCharacteristic)
    {

        NSString *string = [NSString stringWithUTF8String:[[characteristic value] bytes]];
        [self.delegate didReceiveData:string];
    }
    else if ([characteristic.UUID isEqual:self.class.hardwareRevisionStringUUID])
    {
        NSString *hwRevision = @"";
        const uint8_t *bytes = characteristic.value.bytes;
        for (int i = 0; i < characteristic.value.length; i++)
        {
            NSLog(@"%x", bytes[i]);
            hwRevision = [hwRevision stringByAppendingFormat:@"0x%02x, ", bytes[i]];
        }

        [self.delegate didReadHardwareRevisionString:[hwRevision substringToIndex:
            hwRevision.length-2]];
    }

    if (!didSendOnPref){
        [self sendUserPrefInfo];
        if (connected) didSendOnPref = YES;
    }
}
@end
```

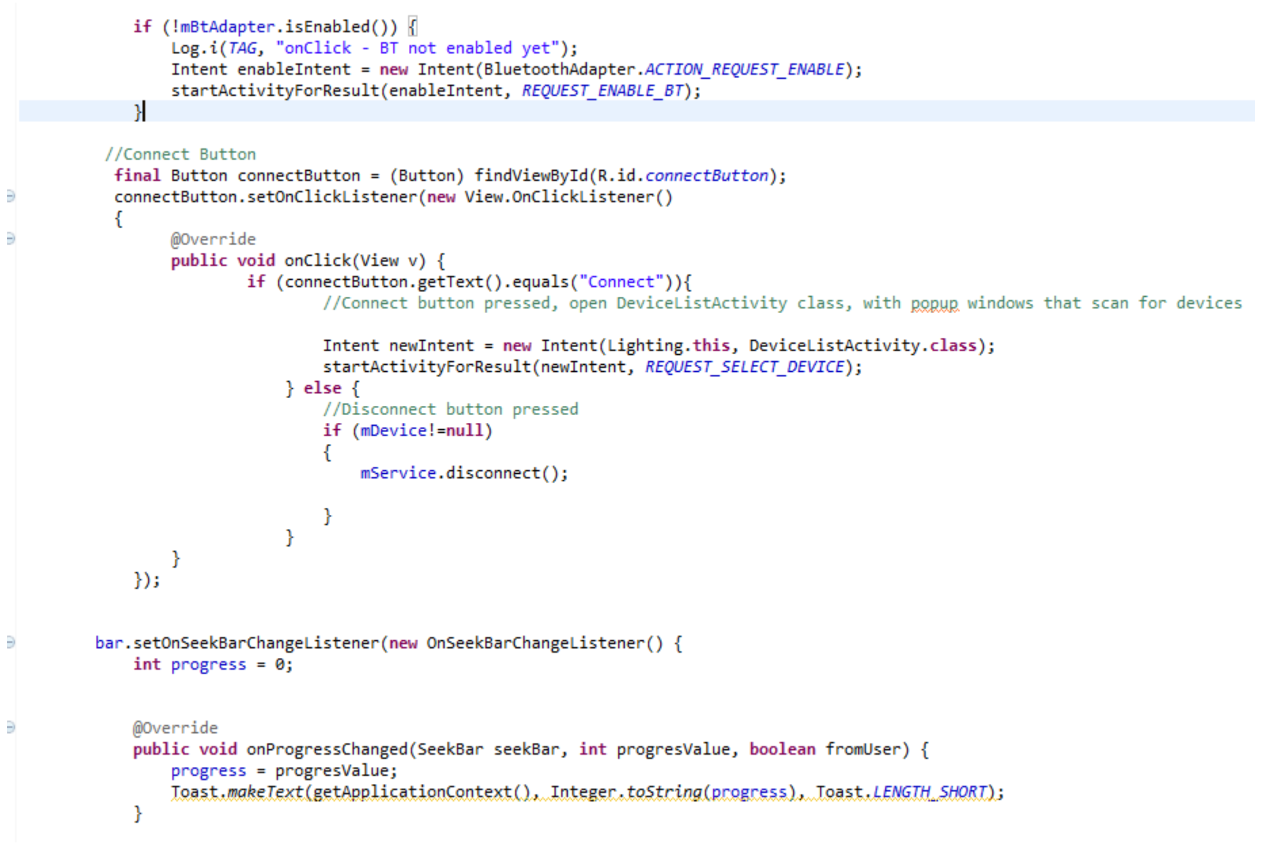
Android Mobile Application



```java
package com.nordicsemi.nrfUARTv2;

import java.io.UnsupportedEncodingException;
import java.text.DateFormat;
import java.util.Date;

import android.app.Activity;
import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.res.Configuration;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.RadioGroup;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.widget.Switch;
import android.widget.TextView;
import android.widget.Toast;

public class Lighting extends Activity {

    SeekBar bar;
    Switch sw;
    private static final int REQUEST_SELECT_DEVICE = 1;
    private static final int REQUEST_ENABLE_BT = 2;
```

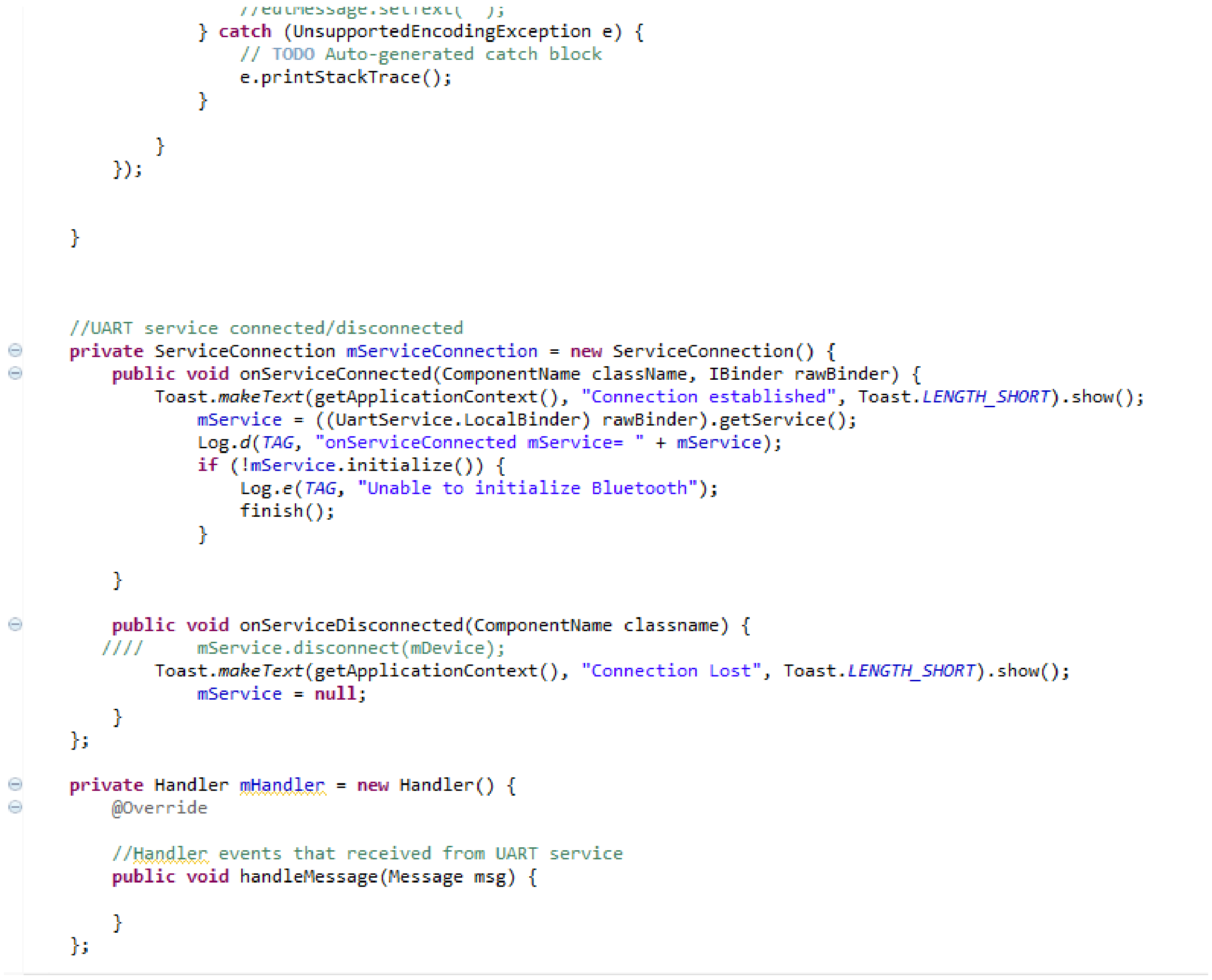


```
private static final int UART_PROFILE_READY = 10;
private BluetoothDevice mDevice = null;
public static final String TAG = "nRFUART";
private static final int UART_PROFILE_CONNECTED = 20;
private static final int UART_PROFILE_DISCONNECTED = 21;
private static final int STATE_OFF = 10;

TextView mRemoteRssiVal;
RadioGroup mRg;
private int mState = UART_PROFILE_DISCONNECTED;
private UartService mService = null;
private BluetoothAdapter mBtAdapter = null;
private ListView messageListView;
private ArrayAdapter<String> listAdapter;
private Button btnConnectDisconnect,btnSend;
private EditText edtMessage;

@Override
  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.lighting);
    Toast.makeText(this, "On Lighting Screen", Toast.LENGTH_SHORT).show();

    Intent intent = this.getIntent();
    Bundle bundle = intent.getExtras();

    //mService =   (UartService)bundle.getSerializable("mService");
    bar = (SeekBar) findViewById(R.id.seekBar1);
    sw = (Switch) findViewById(R.id.switch1);

      service_init();


        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
        if (mBtAdapter == null) {
            Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show();
            finish();
            return;
        }
```



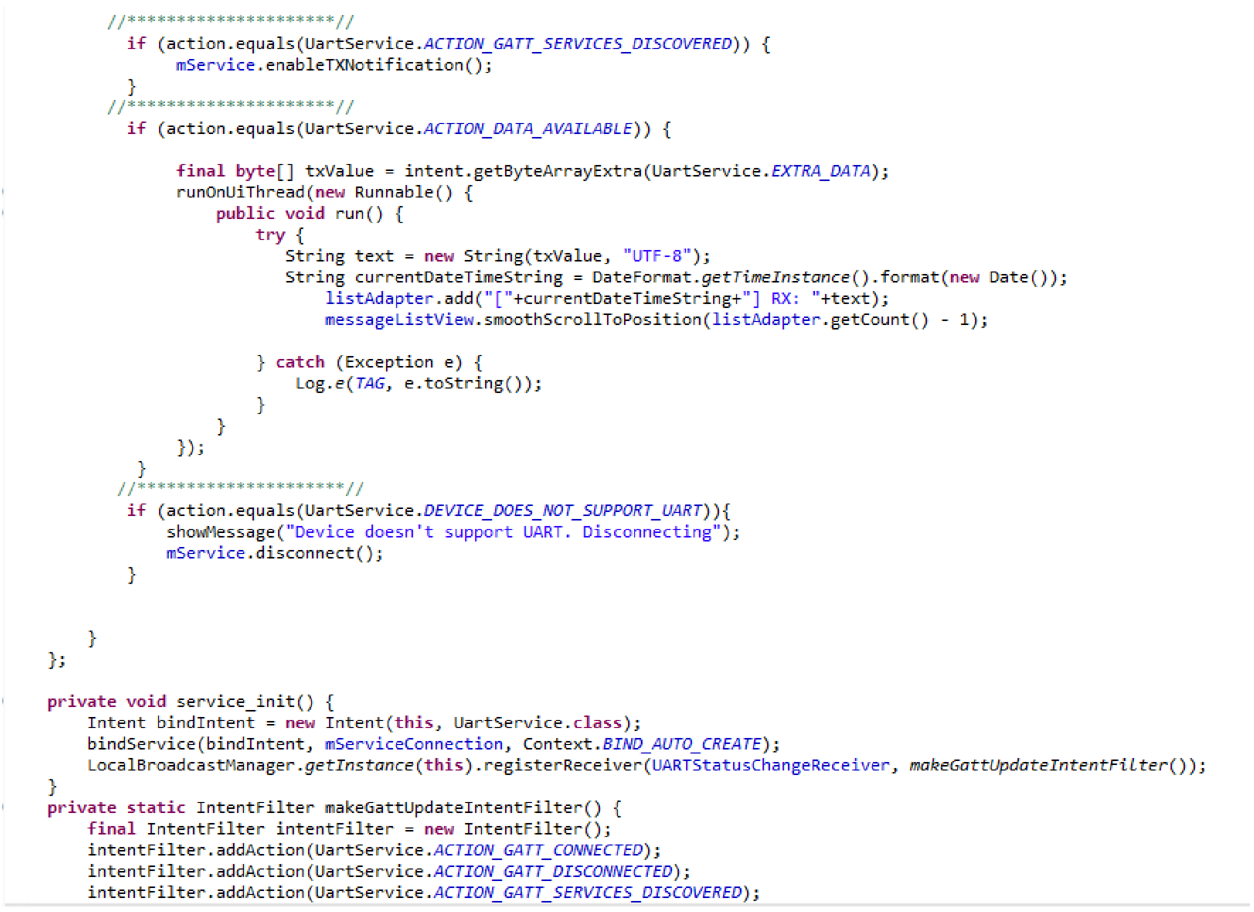
```
            if (!mBtAdapter.isEnabled()) {
                Log.i(TAG, "onClick - BT not enabled yet");
                Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
            }

         //Connect Button
          final Button connectButton = (Button) findViewById(R.id.connectButton);
          connectButton.setOnClickListener(new View.OnClickListener()
          {
                @Override
                public void onClick(View v) {
                        if (connectButton.getText().equals("Connect")){
                                //Connect button pressed, open DeviceListActivity class, with popup windows that scan for devices

                                Intent newIntent = new Intent(Lighting.this, DeviceListActivity.class);
                                startActivityForResult(newIntent, REQUEST_SELECT_DEVICE);
                            } else {
                                //Disconnect button pressed
                                if (mDevice!=null)
                                {
                                    mService.disconnect();

                                }
                            }
                }
            });


        bar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {
            int progress = 0;


            @Override
            public void onProgressChanged(SeekBar seekBar, int progresValue, boolean fromUser) {
                progress = progresValue;
                Toast.makeText(getApplicationContext(), Integer.toString(progress), Toast.LENGTH_SHORT);
            }
```

```
,

@Override
public void onStartTrackingTouch(SeekBar seekBar) {
    // Do something here, if you want to do anything at the start of
    // touching the seekbar
}

@Override
public void onStopTrackingTouch(SeekBar seekBar) {
    //Send data via bluetooth

    //Build the string
    String str = "0";
    if(sw.isChecked())
    {
        str += "1";
    }
    else
    {
        str += "0";
    }
    str += String.format("%03d", progress);
    Toast.makeText(getApplicationContext(), str, Toast.LENGTH_SHORT).show();
    byte[] value;
    try {
        //send data to service
        value = str.getBytes("UTF-8");
        if(mService == null)
        {

            showMessage("mService is null");
        }

        mService.writeRXCharacteristic(value);
        //Update the log with time stamp
        //String currentDateTimeString = DateFormat.getTimeInstance().format(new Date());
        //listAdapter.add("["+currentDateTimeString+"] TX: "+ message);
        //messageListView.smoothScrollToPosition(listAdapter.getCount() - 1);
        //edtMessage.setText("");
```

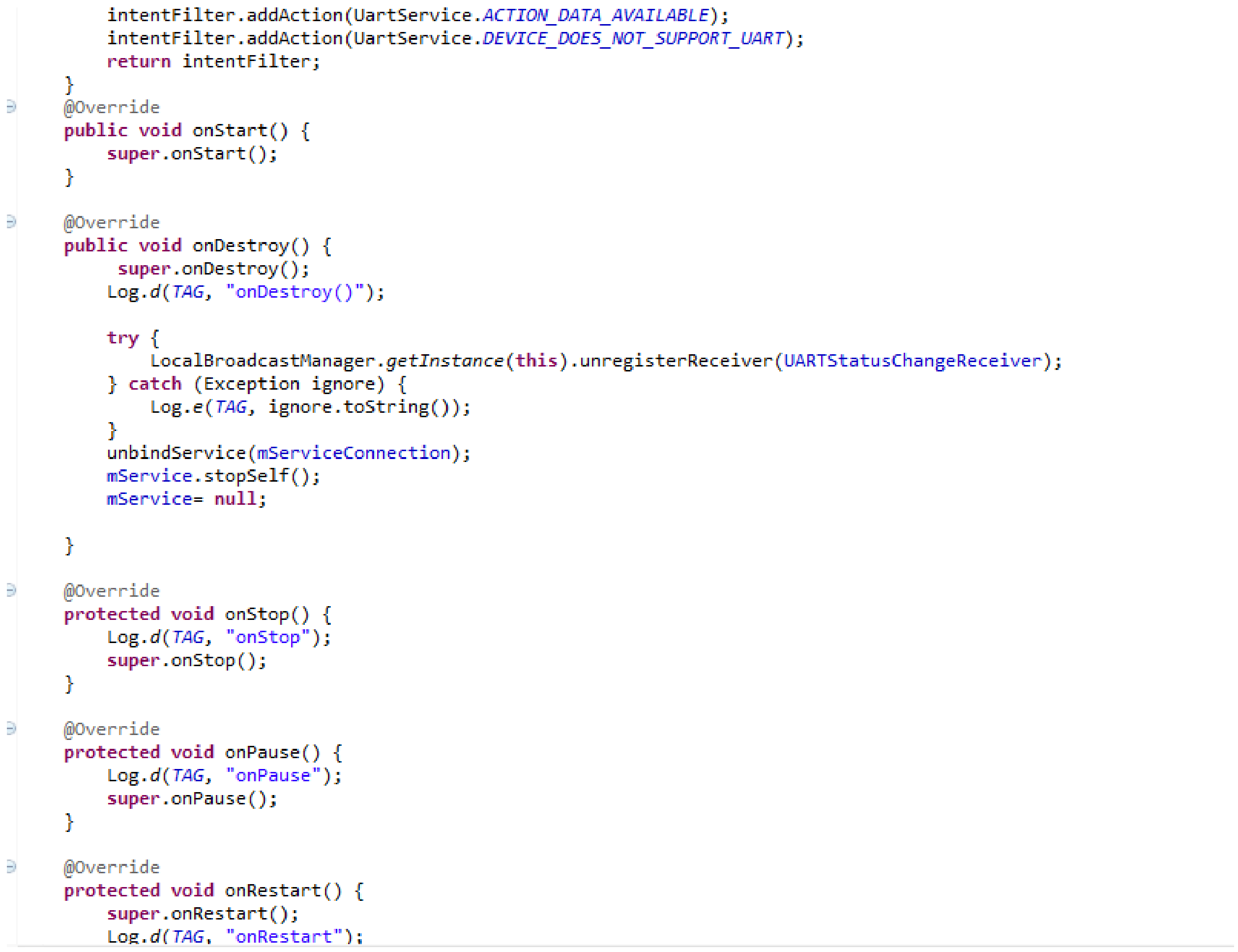


```
                //edtMessage.setText("");
            } catch (UnsupportedEncodingException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }
    });


}



//UART service connected/disconnected
private ServiceConnection mServiceConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder rawBinder) {
        Toast.makeText(getApplicationContext(), "Connection established", Toast.LENGTH_SHORT).show();
            mService = ((UartService.LocalBinder) rawBinder).getService();
            Log.d(TAG, "onServiceConnected mService= " + mService);
            if (!mService.initialize()) {
                Log.e(TAG, "Unable to initialize Bluetooth");
                finish();
            }

    }

    public void onServiceDisconnected(ComponentName classname) {
   ////     mService.disconnect(mDevice);
        Toast.makeText(getApplicationContext(), "Connection Lost", Toast.LENGTH_SHORT).show();
            mService = null;
    }
};

private Handler mHandler = new Handler() {
    @Override

    //Handler events that received from UART service
    public void handleMessage(Message msg) {

    }
};
```

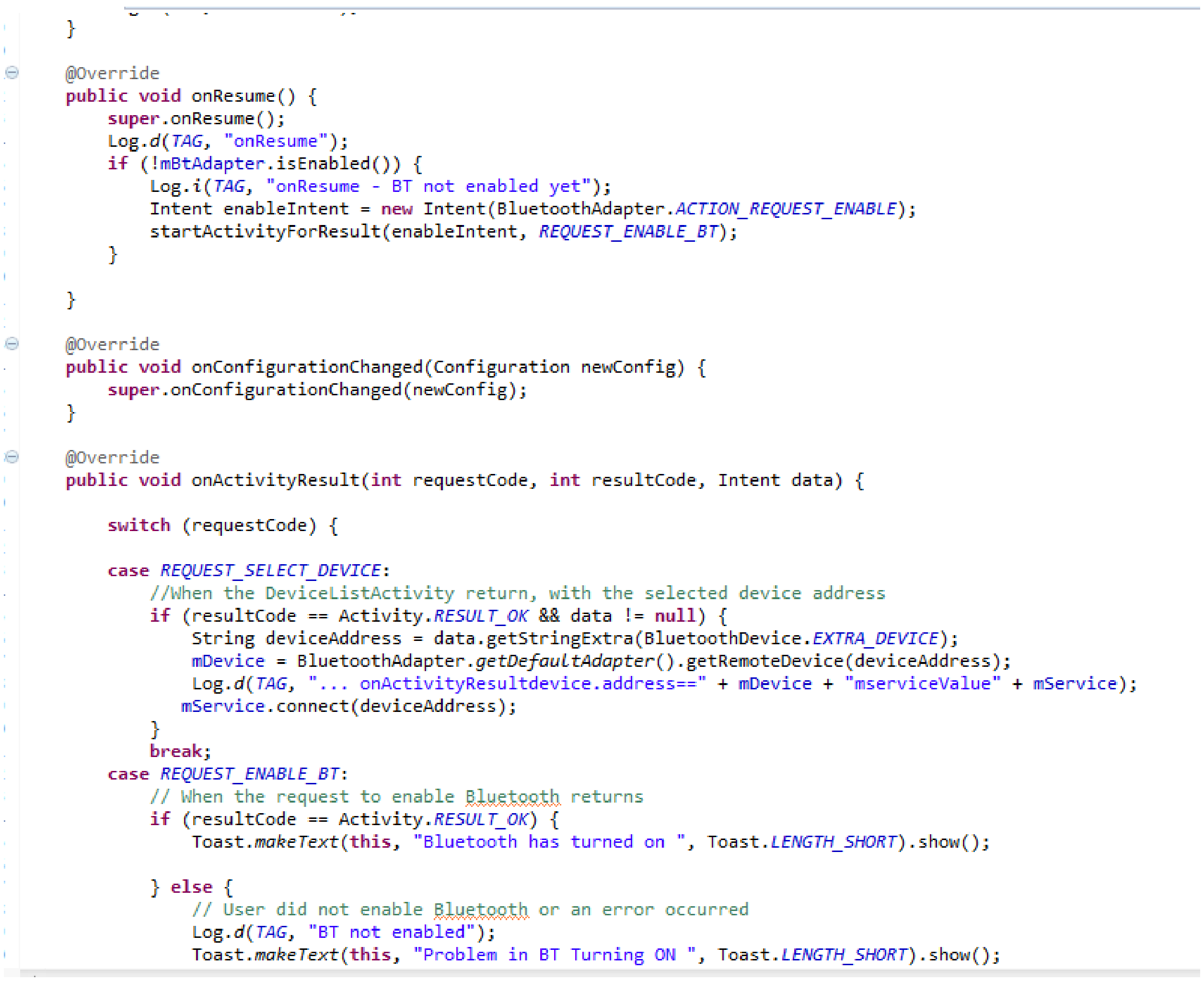
```java
private final BroadcastReceiver UARTStatusChangeReceiver = new BroadcastReceiver() {

    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        final Intent mIntent = intent;
       //*********************//
        if (action.equals(UartService.ACTION_GATT_CONNECTED)) {
             runOnUiThread(new Runnable() {
                 public void run() {
                        String currentDateTimeString = DateFormat.getTimeInstance().format(new Date());
                         Log.d(TAG, "UART_CONNECT_MSG");
                         btnConnectDisconnect.setText("Disconnect");
                         edtMessage.setEnabled(true);
                         btnSend.setEnabled(true);
                         ((TextView) findViewById(R.id.deviceName)).setText(mDevice.getName()+ " - ready");
                         listAdapter.add("["+currentDateTimeString+"] Connected to: "+ mDevice.getName());
                            messageListView.smoothScrollToPosition(listAdapter.getCount() - 1);
                         mState = UART_PROFILE_CONNECTED;
                 }
             });
        }

      //*********************//
        if (action.equals(UartService.ACTION_GATT_DISCONNECTED)) {
             runOnUiThread(new Runnable() {
                 public void run() {
                         String currentDateTimeString = DateFormat.getTimeInstance().format(new Date());
                         Log.d(TAG, "UART_DISCONNECT_MSG");
                         btnConnectDisconnect.setText("Connect");
                         edtMessage.setEnabled(false);
                         btnSend.setEnabled(false);
                         ((TextView) findViewById(R.id.deviceName)).setText("Not Connected");
                         listAdapter.add("["+currentDateTimeString+"] Disconnected to: "+ mDevice.getName());
                         mState = UART_PROFILE_DISCONNECTED;
                         mService.close();
                        //setUiState();

                 }
             });
        }
```



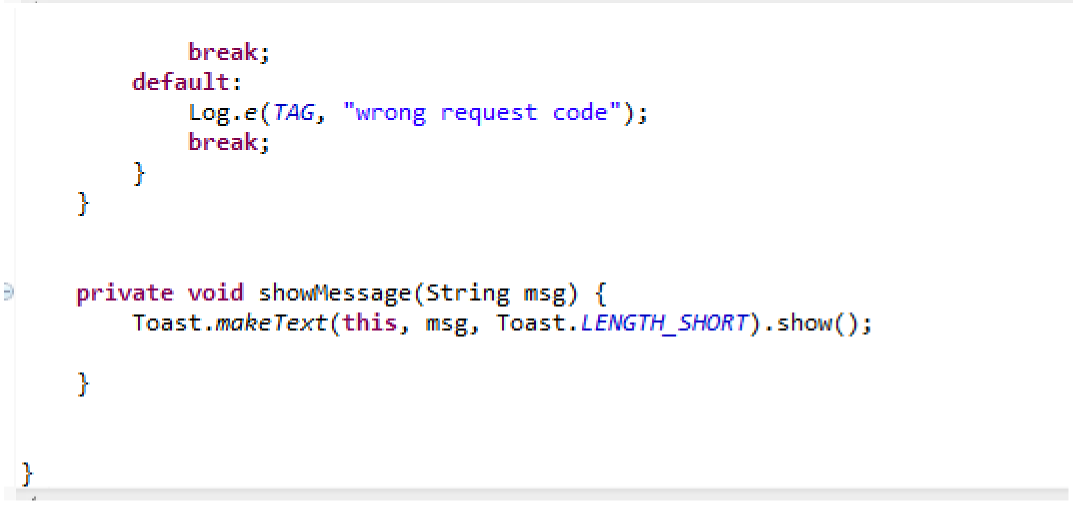
```
      //*********************//
        if (action.equals(UartService.ACTION_GATT_SERVICES_DISCOVERED)) {
             mService.enableTXNotification();
        }
      //*********************//
        if (action.equals(UartService.ACTION_DATA_AVAILABLE)) {

             final byte[] txValue = intent.getByteArrayExtra(UartService.EXTRA_DATA);
             runOnUiThread(new Runnable() {
                 public void run() {
                     try {
                        String text = new String(txValue, "UTF-8");
                        String currentDateTimeString = DateFormat.getTimeInstance().format(new Date());
                            listAdapter.add("["+currentDateTimeString+"] RX: "+text);
                            messageListView.smoothScrollToPosition(listAdapter.getCount() - 1);

                     } catch (Exception e) {
                         Log.e(TAG, e.toString());
                     }
                 }
             });
         }
       //*********************//
        if (action.equals(UartService.DEVICE_DOES_NOT_SUPPORT_UART)){
            showMessage("Device doesn't support UART. Disconnecting");
            mService.disconnect();
        }


    }
};

private void service_init() {
    Intent bindIntent = new Intent(this, UartService.class);
    bindService(bindIntent, mServiceConnection, Context.BIND_AUTO_CREATE);
    LocalBroadcastManager.getInstance(this).registerReceiver(UARTStatusChangeReceiver, makeGattUpdateIntentFilter());
}
private static IntentFilter makeGattUpdateIntentFilter() {
    final IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(UartService.ACTION_GATT_CONNECTED);
    intentFilter.addAction(UartService.ACTION_GATT_DISCONNECTED);
    intentFilter.addAction(UartService.ACTION_GATT_SERVICES_DISCOVERED);
```



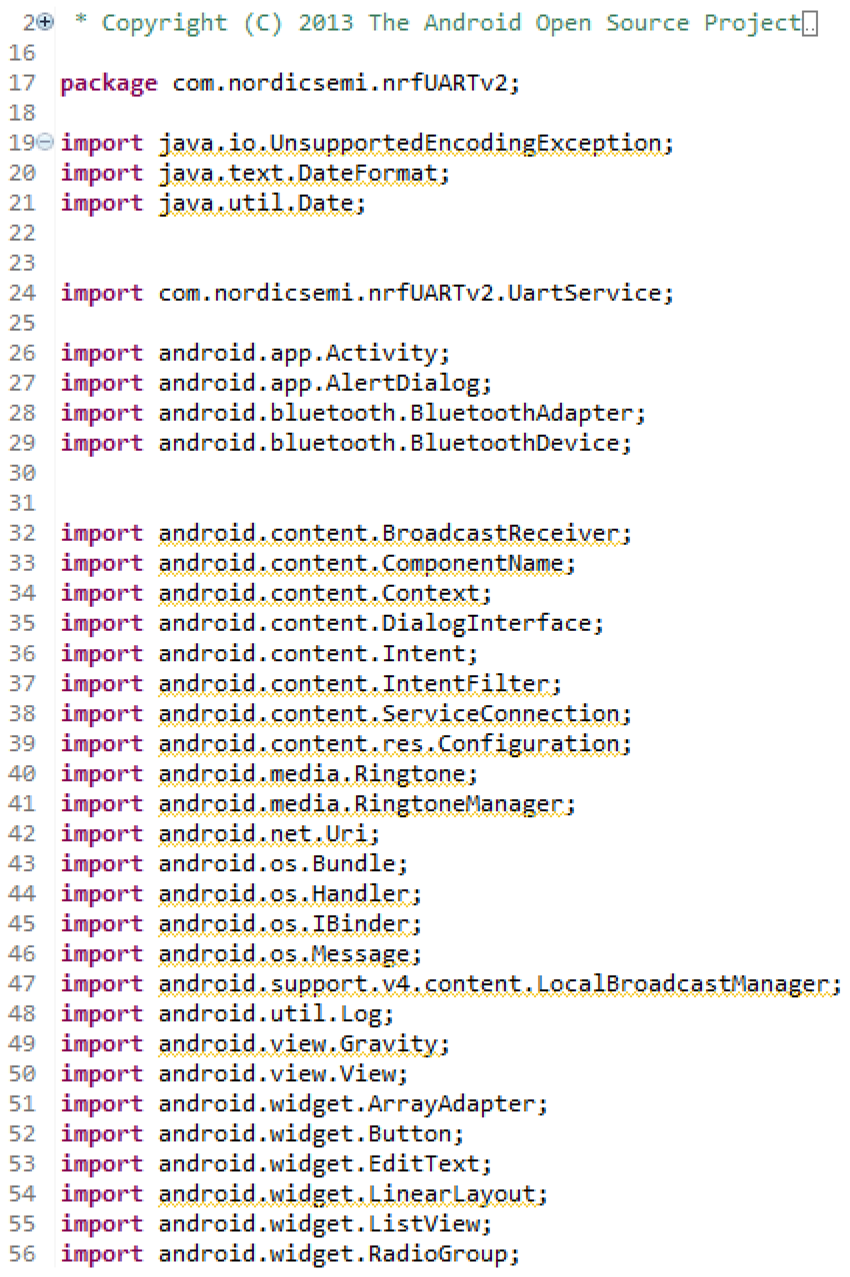
```
        intentFilter.addAction(UartService.ACTION_DATA_AVAILABLE);
        intentFilter.addAction(UartService.DEVICE_DOES_NOT_SUPPORT_UART);
        return intentFilter;
    }
    @Override
    public void onStart() {
        super.onStart();
    }

    @Override
    public void onDestroy() {
         super.onDestroy();
        Log.d(TAG, "onDestroy()");

        try {
            LocalBroadcastManager.getInstance(this).unregisterReceiver(UARTStatusChangeReceiver);
        } catch (Exception ignore) {
            Log.e(TAG, ignore.toString());
        }
        unbindService(mServiceConnection);
        mService.stopSelf();
        mService= null;

    }

    @Override
    protected void onStop() {
        Log.d(TAG, "onStop");
        super.onStop();
    }

    @Override
    protected void onPause() {
        Log.d(TAG, "onPause");
        super.onPause();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d(TAG, "onRestart");
```



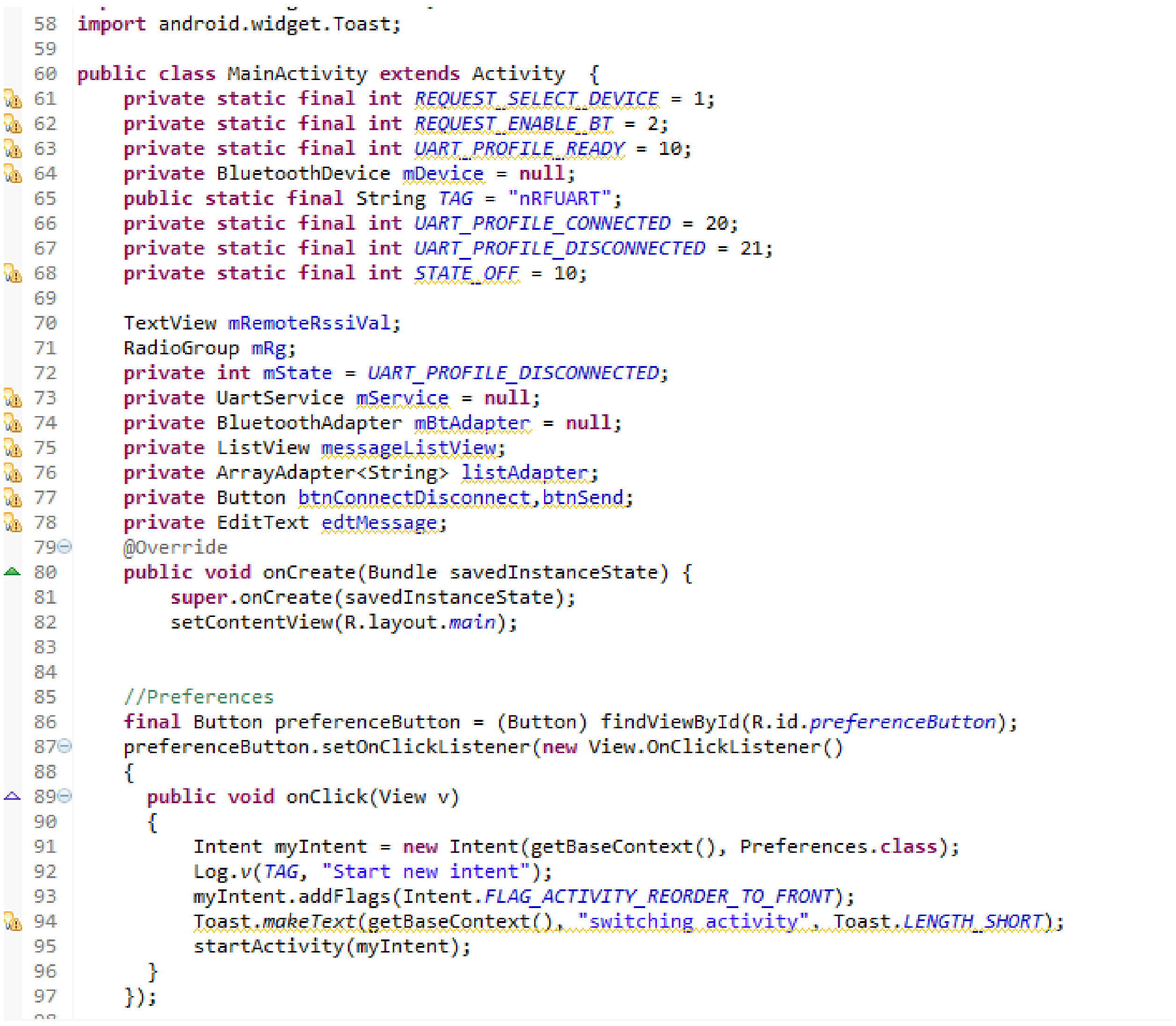
```java
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
        if (!mBtAdapter.isEnabled()) {
            Log.i(TAG, "onResume - BT not enabled yet");
            Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
        }

    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);
    }

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

        switch (requestCode) {

        case REQUEST_SELECT_DEVICE:
            //When the DeviceListActivity return, with the selected device address
            if (resultCode == Activity.RESULT_OK && data != null) {
                String deviceAddress = data.getStringExtra(BluetoothDevice.EXTRA_DEVICE);
                mDevice = BluetoothAdapter.getDefaultAdapter().getRemoteDevice(deviceAddress);
                Log.d(TAG, "... onActivityResultdevice.address==" + mDevice + "mserviceValue" + mService);
               mService.connect(deviceAddress);
            }
            break;
        case REQUEST_ENABLE_BT:
            // When the request to enable Bluetooth returns
            if (resultCode == Activity.RESULT_OK) {
                Toast.makeText(this, "Bluetooth has turned on ", Toast.LENGTH_SHORT).show();

            } else {
                // User did not enable Bluetooth or an error occurred
                Log.d(TAG, "BT not enabled");
                Toast.makeText(this, "Problem in BT Turning ON ", Toast.LENGTH_SHORT).show();
```

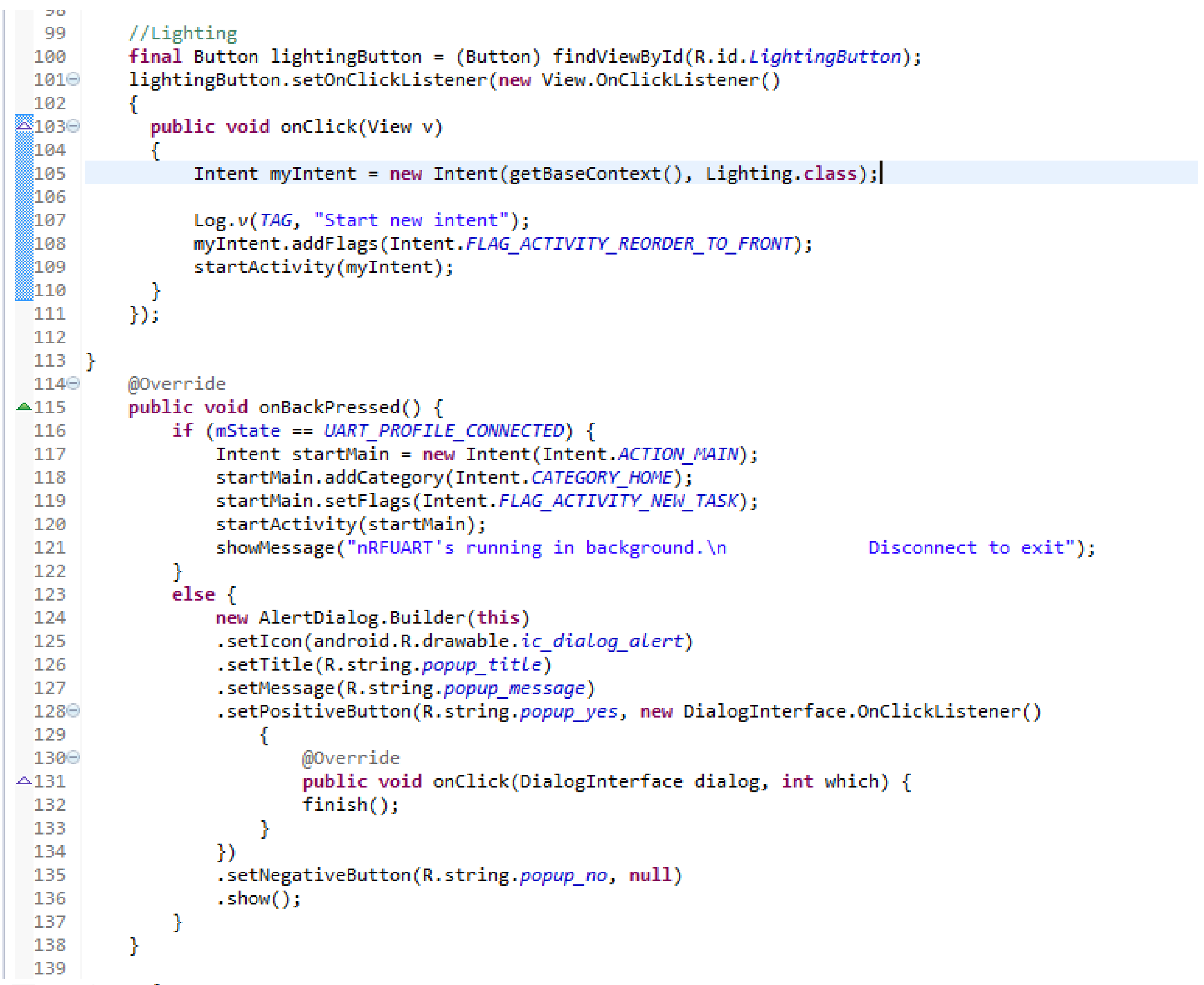


```java
            break;
        default:
            Log.e(TAG, "wrong request code");
            break;
        }
    }


    private void showMessage(String msg) {
        Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();

    }


}
```

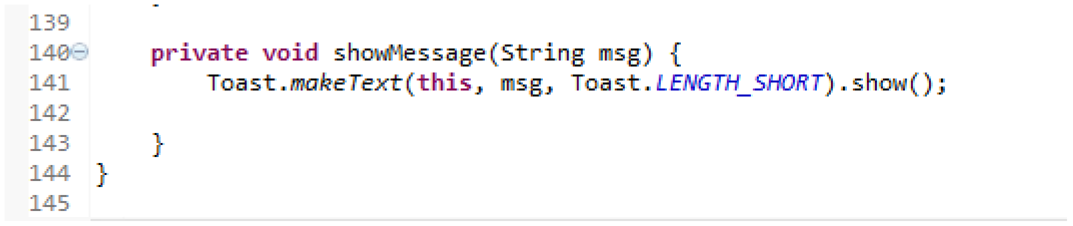


```
 * Copyright (C) 2013 The Android Open Source Project

package com.nordicsemi.nrfUARTv2;

import java.io.UnsupportedEncodingException;
import java.text.DateFormat;
import java.util.Date;


import com.nordicsemi.nrfUARTv2.UartService;

import android.app.Activity;
import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;


import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.res.Configuration;
import android.media.Ringtone;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
import android.view.Gravity;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.RadioGroup;
```

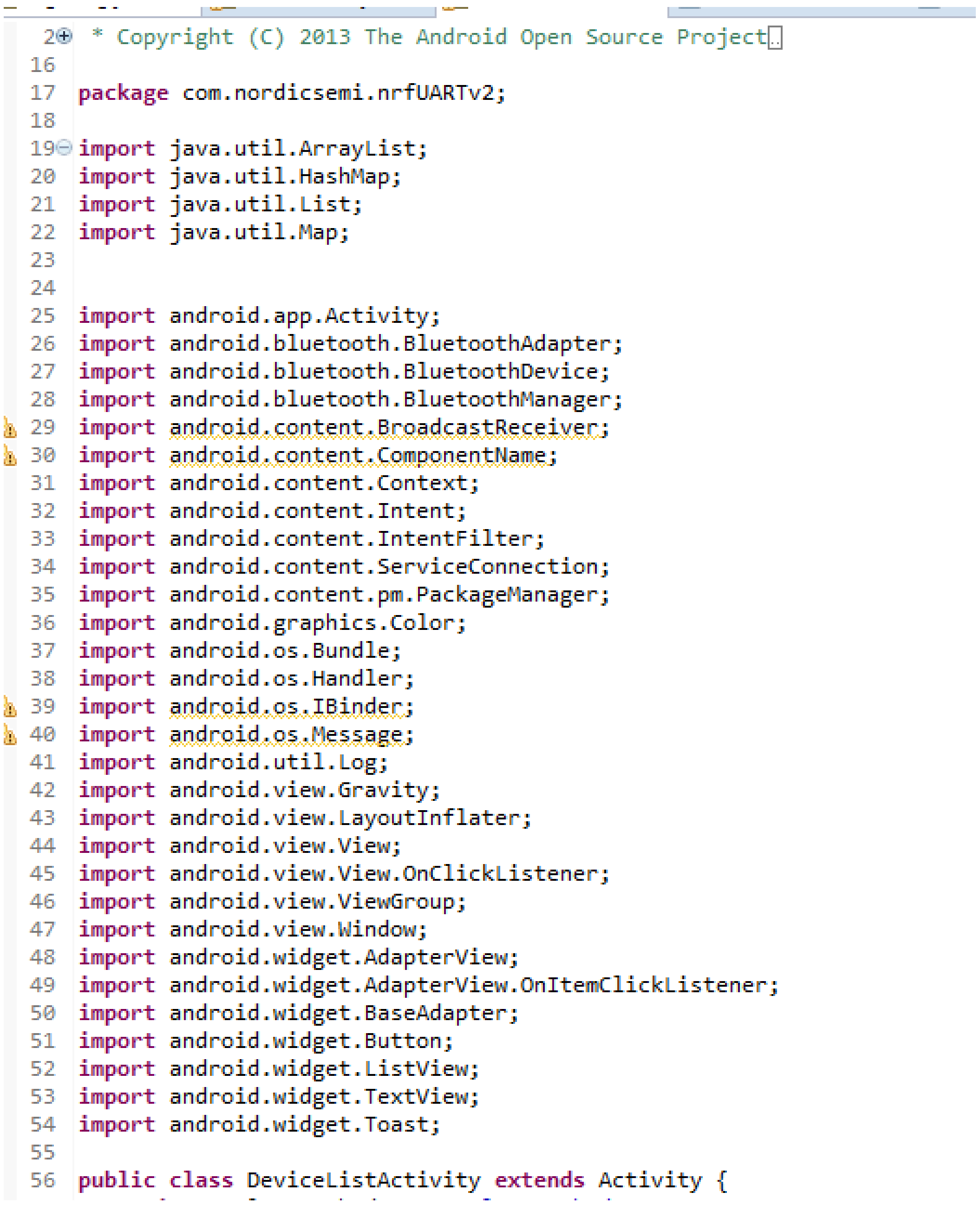

```
import android.widget.Toast;

public class MainActivity extends Activity  {
    private static final int REQUEST_SELECT_DEVICE = 1;
    private static final int REQUEST_ENABLE_BT = 2;
    private static final int UART_PROFILE_READY = 10;
    private BluetoothDevice mDevice = null;
    public static final String TAG = "nRFUART";
    private static final int UART_PROFILE_CONNECTED = 20;
    private static final int UART_PROFILE_DISCONNECTED = 21;
    private static final int STATE_OFF = 10;

    TextView mRemoteRssiVal;
    RadioGroup mRg;
    private int mState = UART_PROFILE_DISCONNECTED;
    private UartService mService = null;
    private BluetoothAdapter mBtAdapter = null;
    private ListView messageListView;
    private ArrayAdapter<String> listAdapter;
    private Button btnConnectDisconnect,btnSend;
    private EditText edtMessage;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);


    //Preferences
    final Button preferenceButton = (Button) findViewById(R.id.preferenceButton);
    preferenceButton.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View v)
      {
          Intent myIntent = new Intent(getBaseContext(), Preferences.class);
          Log.v(TAG, "Start new intent");
          myIntent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
          Toast.makeText(getBaseContext(), "switching activity", Toast.LENGTH_SHORT);
          startActivity(myIntent);
      }
    });
```



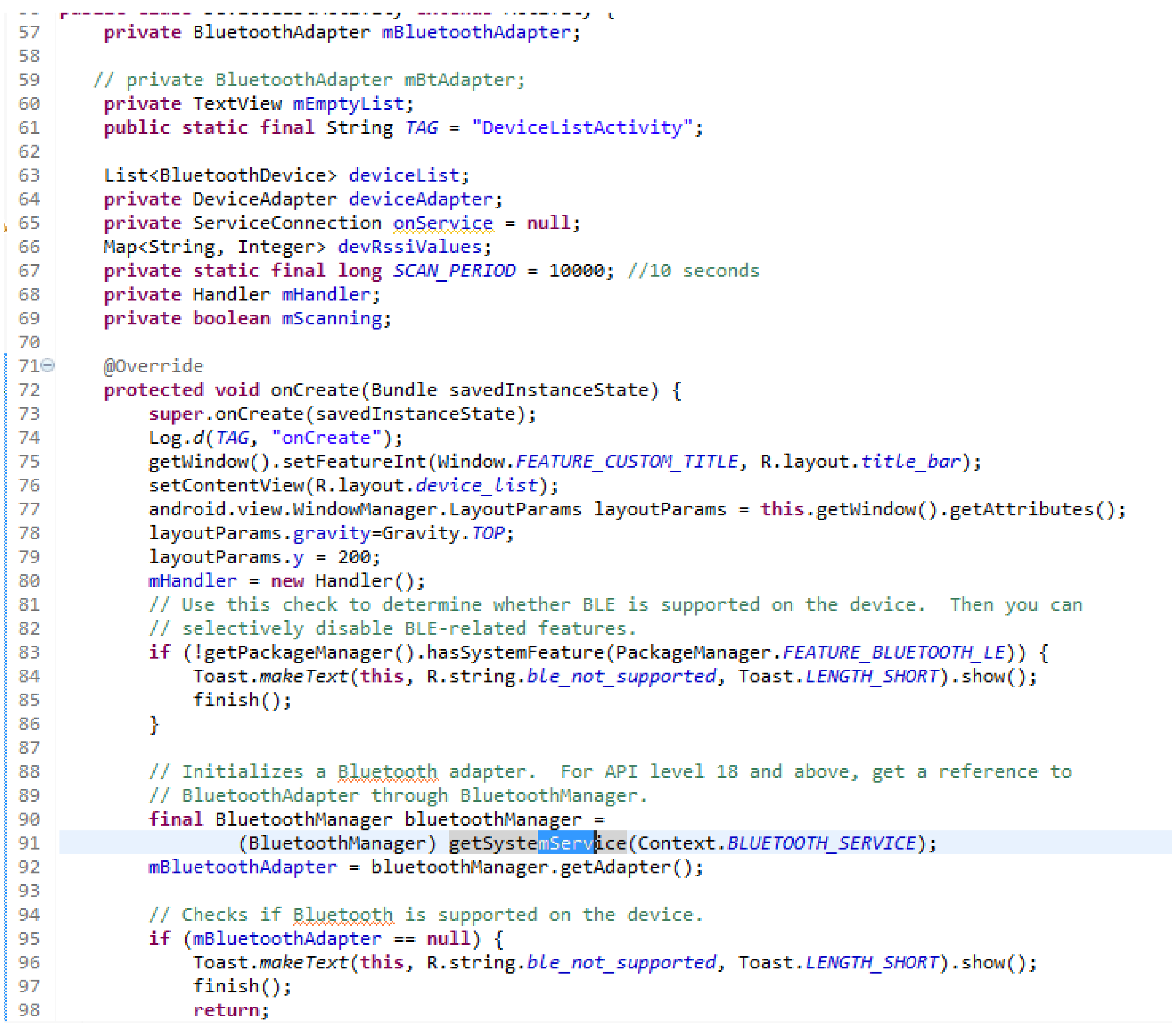
```java
    //Lighting
    final Button lightingButton = (Button) findViewById(R.id.LightingButton);
    lightingButton.setOnClickListener(new View.OnClickListener()
    {
      public void onClick(View v)
      {
          Intent myIntent = new Intent(getBaseContext(), Lighting.class);

          Log.v(TAG, "Start new intent");
          myIntent.addFlags(Intent.FLAG_ACTIVITY_REORDER_TO_FRONT);
          startActivity(myIntent);
      }
    });

}
    @Override
    public void onBackPressed() {
        if (mState == UART_PROFILE_CONNECTED) {
            Intent startMain = new Intent(Intent.ACTION_MAIN);
            startMain.addCategory(Intent.CATEGORY_HOME);
            startMain.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(startMain);
            showMessage("nRFUART's running in background.\n             Disconnect to exit");
        }
        else {
            new AlertDialog.Builder(this)
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setTitle(R.string.popup_title)
            .setMessage(R.string.popup_message)
            .setPositiveButton(R.string.popup_yes, new DialogInterface.OnClickListener()
                {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                    finish();
                }
            })
            .setNegativeButton(R.string.popup_no, null)
            .show();
        }
    }
```



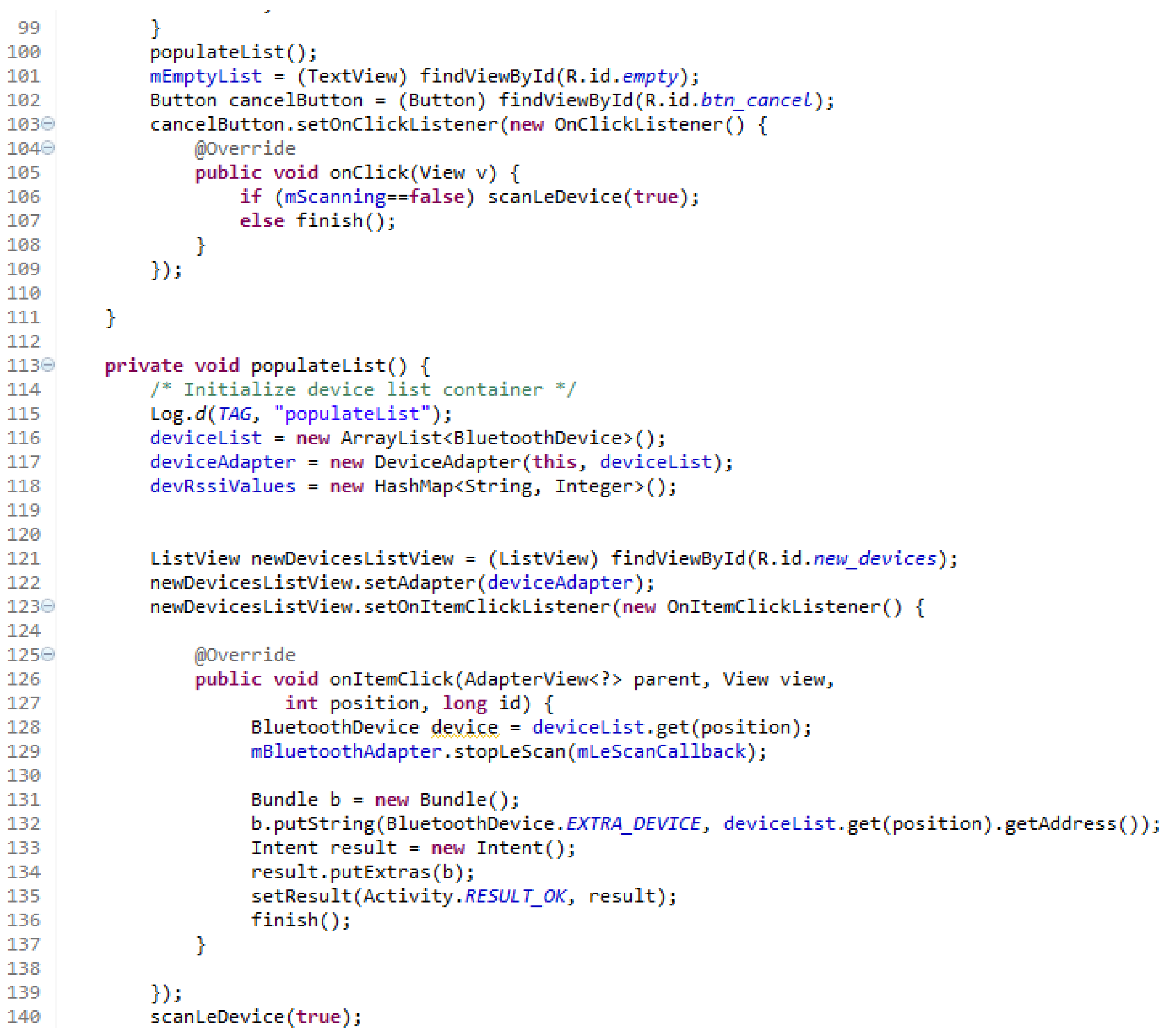
```java

    private void showMessage(String msg) {
        Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();

    }
}

```



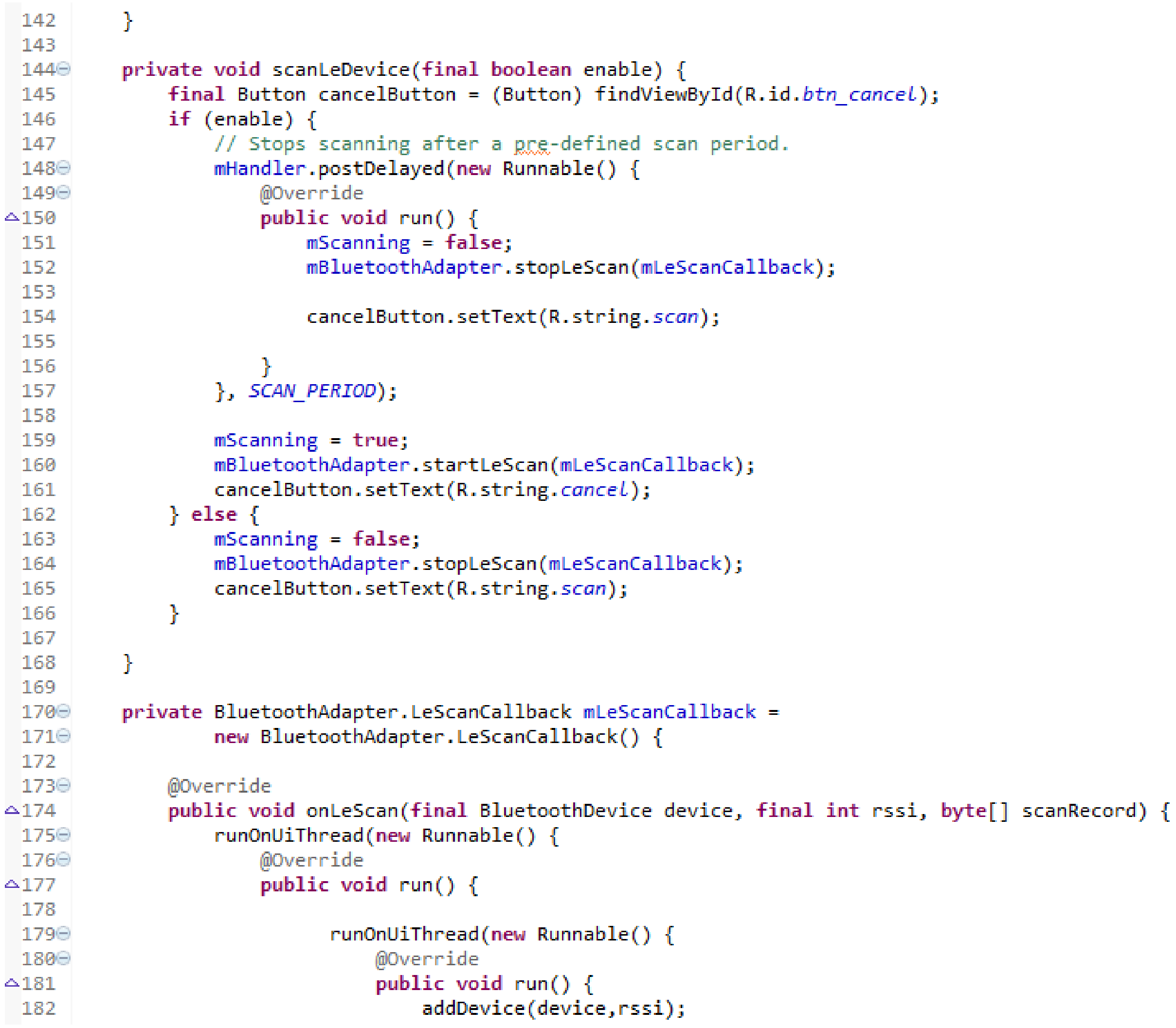
```
 * Copyright (C) 2013 The Android Open Source Project

package com.nordicsemi.nrfUARTv2;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.util.Log;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.view.Window;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.BaseAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class DeviceListActivity extends Activity {
```



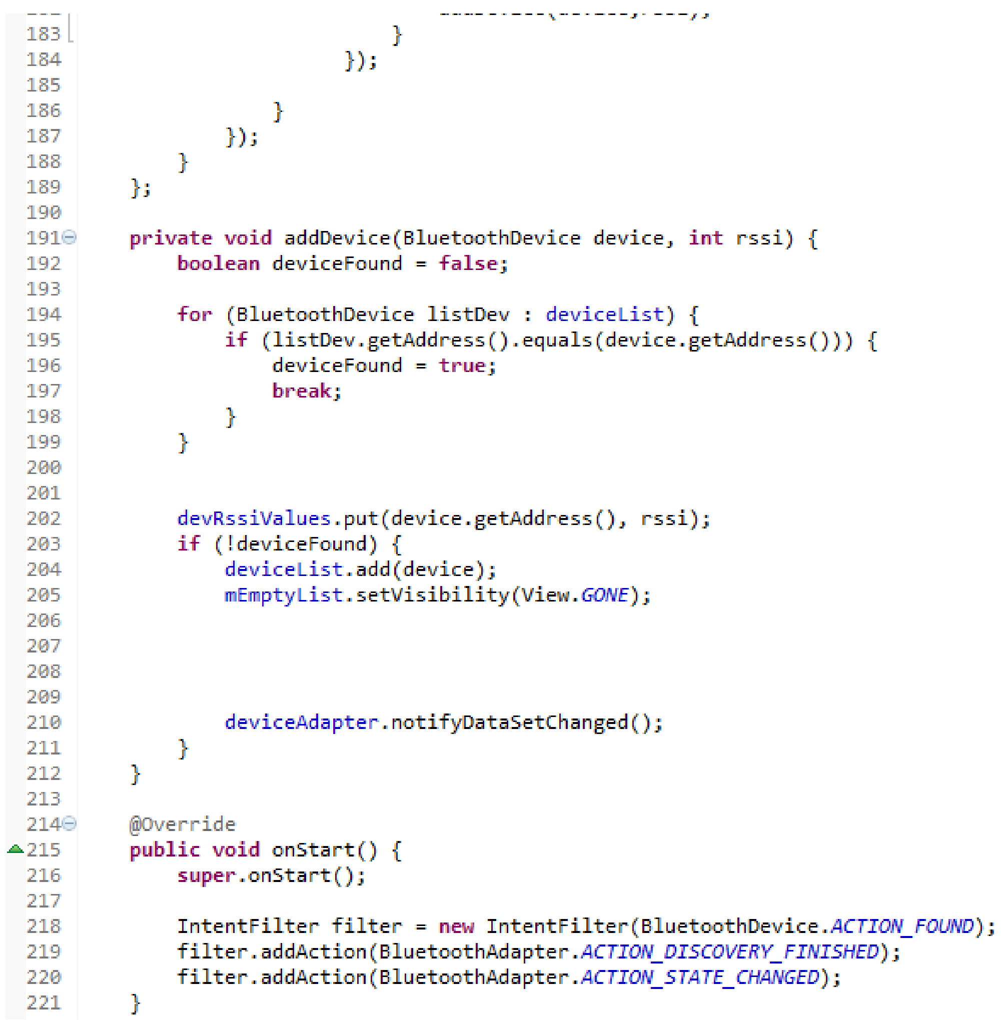
```java
    private BluetoothAdapter mBluetoothAdapter;

   // private BluetoothAdapter mBtAdapter;
    private TextView mEmptyList;
    public static final String TAG = "DeviceListActivity";

    List<BluetoothDevice> deviceList;
    private DeviceAdapter deviceAdapter;
    private ServiceConnection onService = null;
    Map<String, Integer> devRssiValues;
    private static final long SCAN_PERIOD = 10000; //10 seconds
    private Handler mHandler;
    private boolean mScanning;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate");
        getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.title_bar);
        setContentView(R.layout.device_list);
        android.view.WindowManager.LayoutParams layoutParams = this.getWindow().getAttributes();
        layoutParams.gravity=Gravity.TOP;
        layoutParams.y = 200;
        mHandler = new Handler();
        // Use this check to determine whether BLE is supported on the device.  Then you can
        // selectively disable BLE-related features.
        if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {
            Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).show();
            finish();
        }

        // Initializes a Bluetooth adapter.  For API level 18 and above, get a reference to
        // BluetoothAdapter through BluetoothManager.
        final BluetoothManager bluetoothManager =
                (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
        mBluetoothAdapter = bluetoothManager.getAdapter();

        // Checks if Bluetooth is supported on the device.
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, R.string.ble_not_supported, Toast.LENGTH_SHORT).show();
            finish();
            return;
```



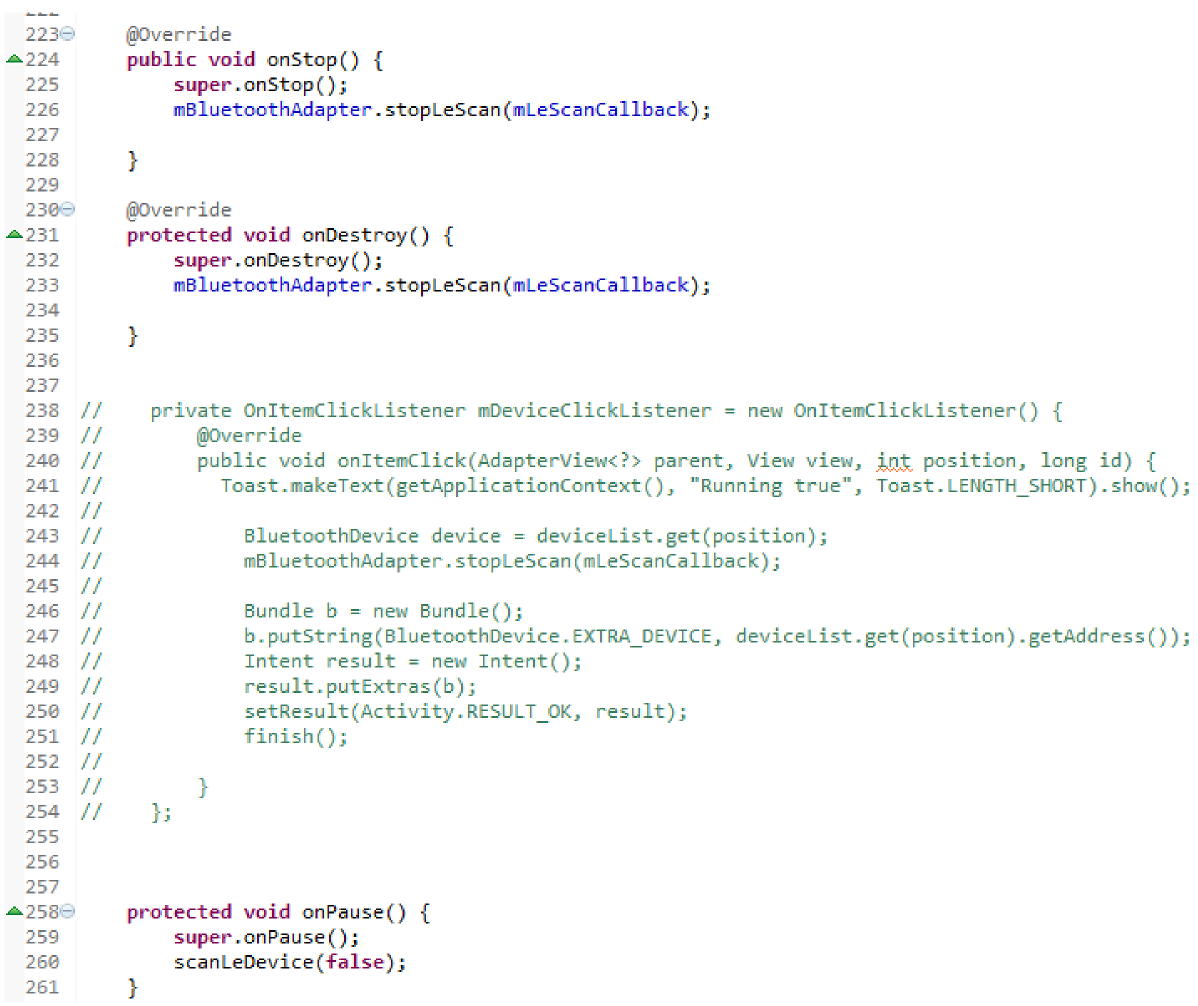
```java
        }
        populateList();
        mEmptyList = (TextView) findViewById(R.id.empty);
        Button cancelButton = (Button) findViewById(R.id.btn_cancel);
        cancelButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mScanning==false) scanLeDevice(true);
                else finish();
            }
        });

    }

    private void populateList() {
        /* Initialize device list container */
        Log.d(TAG, "populateList");
        deviceList = new ArrayList<BluetoothDevice>();
        deviceAdapter = new DeviceAdapter(this, deviceList);
        devRssiValues = new HashMap<String, Integer>();


        ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
        newDevicesListView.setAdapter(deviceAdapter);
        newDevicesListView.setOnItemClickListener(new OnItemClickListener() {

            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id) {
                 BluetoothDevice device = deviceList.get(position);
                 mBluetoothAdapter.stopLeScan(mLeScanCallback);

                 Bundle b = new Bundle();
                 b.putString(BluetoothDevice.EXTRA_DEVICE, deviceList.get(position).getAddress());
                 Intent result = new Intent();
                 result.putExtras(b);
                 setResult(Activity.RESULT_OK, result);
                 finish();
            }

        });
        scanLeDevice(true);
```



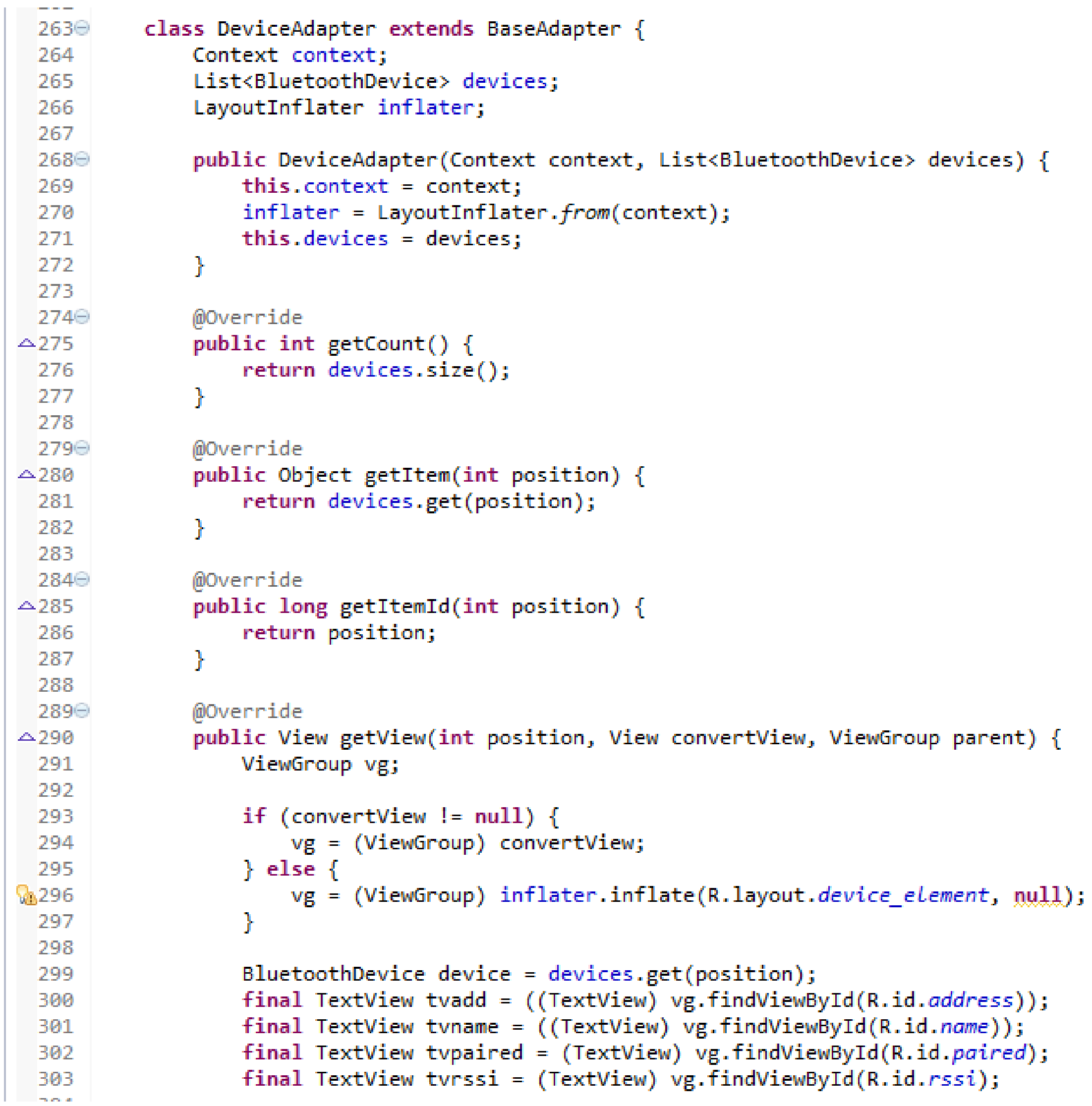
```
    }

    private void scanLeDevice(final boolean enable) {
        final Button cancelButton = (Button) findViewById(R.id.btn_cancel);
        if (enable) {
            // Stops scanning after a pre-defined scan period.
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    mScanning = false;
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);

                    cancelButton.setText(R.string.scan);

                }
            }, SCAN_PERIOD);

            mScanning = true;
            mBluetoothAdapter.startLeScan(mLeScanCallback);
            cancelButton.setText(R.string.cancel);
        } else {
            mScanning = false;
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
            cancelButton.setText(R.string.scan);
        }

    }

    private BluetoothAdapter.LeScanCallback mLeScanCallback =
            new BluetoothAdapter.LeScanCallback() {

        @Override
        public void onLeScan(final BluetoothDevice device, final int rssi, byte[] scanRecord) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {

                      runOnUiThread(new Runnable() {
                          @Override
                          public void run() {
                              addDevice(device,rssi);
```



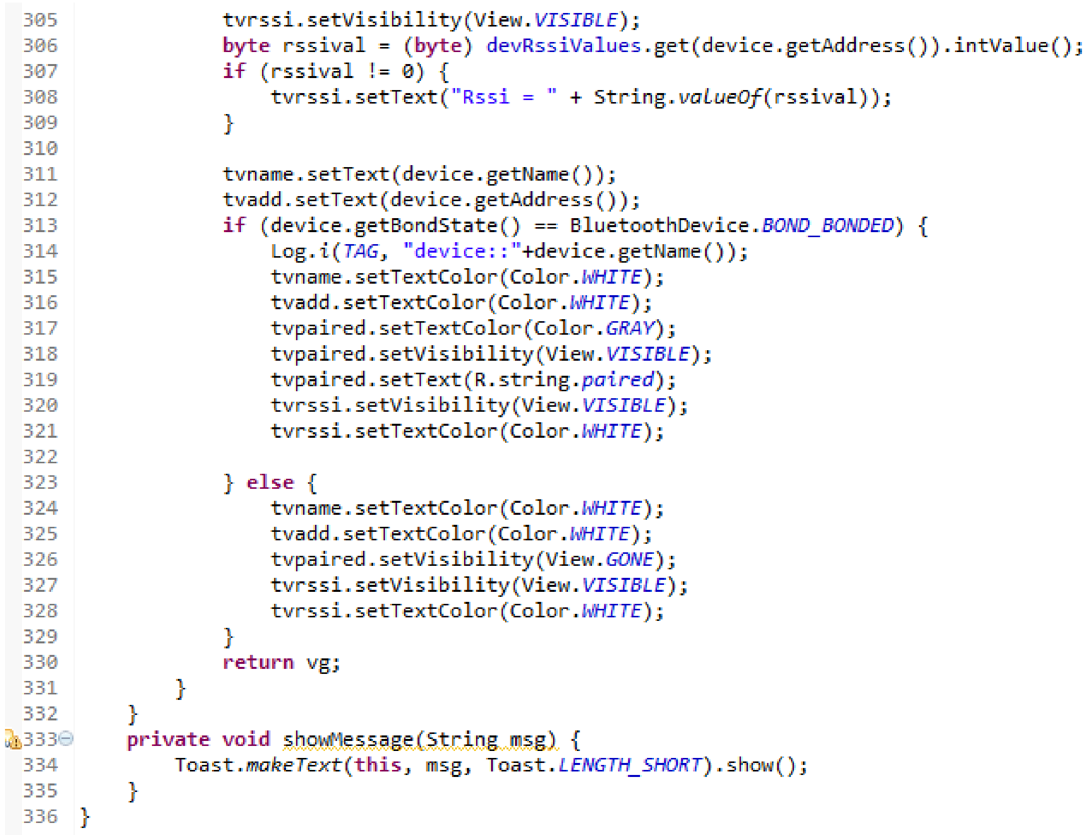
```
                        }
                    });

                }
            });
        }
    };

    private void addDevice(BluetoothDevice device, int rssi) {
        boolean deviceFound = false;

        for (BluetoothDevice listDev : deviceList) {
            if (listDev.getAddress().equals(device.getAddress())) {
                deviceFound = true;
                break;
            }
        }


        devRssiValues.put(device.getAddress(), rssi);
        if (!deviceFound) {
            deviceList.add(device);
            mEmptyList.setVisibility(View.GONE);




            deviceAdapter.notifyDataSetChanged();
        }
    }

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

        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        filter.addAction(BluetoothAdapter.ACTION_STATE_CHANGED);
    }
```



```java
    @Override
    public void onStop() {
        super.onStop();
        mBluetoothAdapter.stopLeScan(mLeScanCallback);

    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mBluetoothAdapter.stopLeScan(mLeScanCallback);

    }


//    private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
//        @Override
//        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
//          Toast.makeText(getApplicationContext(), "Running true", Toast.LENGTH_SHORT).show();
//
//            BluetoothDevice device = deviceList.get(position);
//            mBluetoothAdapter.stopLeScan(mLeScanCallback);
//
//            Bundle b = new Bundle();
//            b.putString(BluetoothDevice.EXTRA_DEVICE, deviceList.get(position).getAddress());
//            Intent result = new Intent();
//            result.putExtras(b);
//            setResult(Activity.RESULT_OK, result);
//            finish();
//
//        }
//    };



    protected void onPause() {
        super.onPause();
        scanLeDevice(false);
    }
```



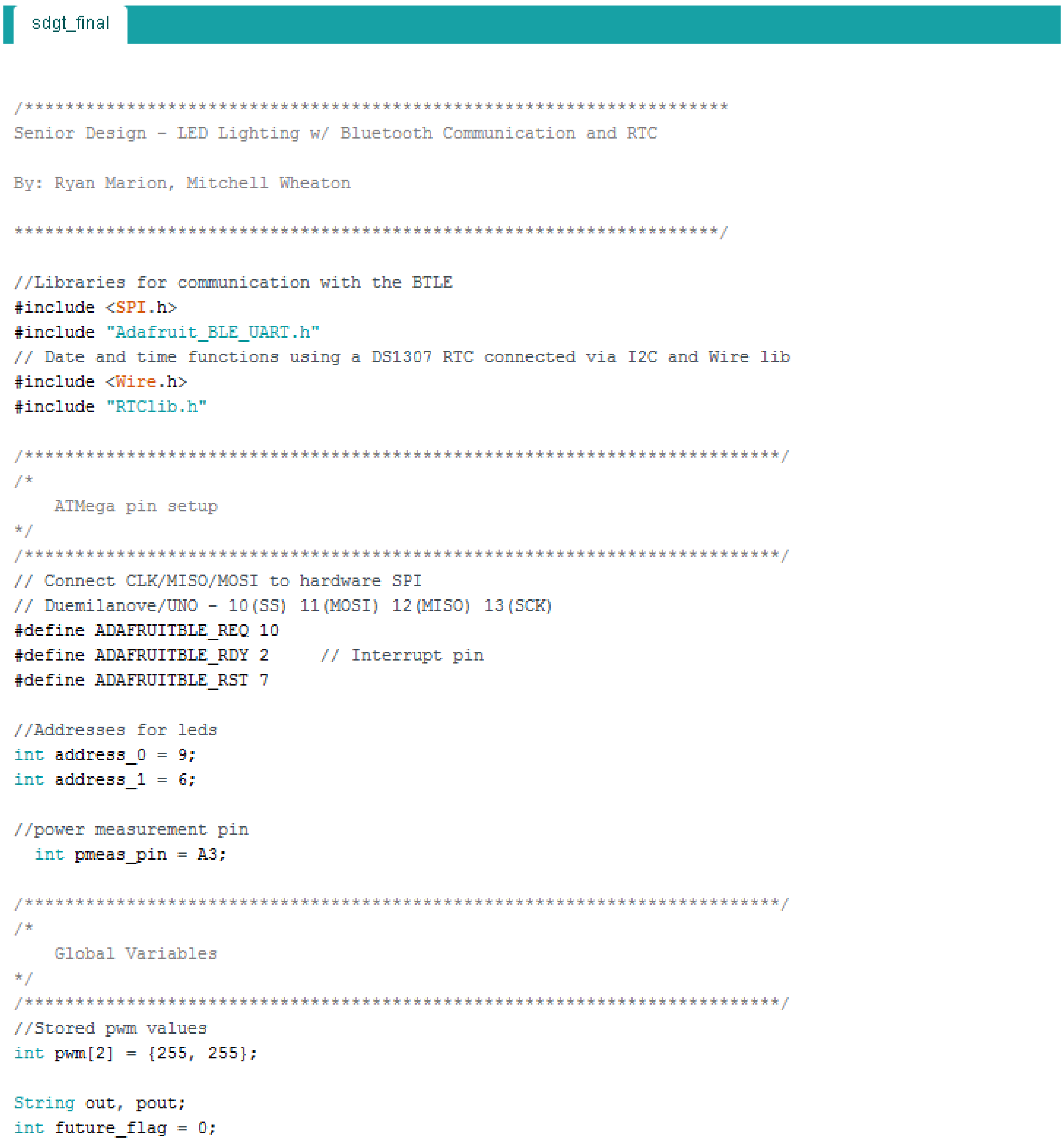
```
    class DeviceAdapter extends BaseAdapter {
        Context context;
        List<BluetoothDevice> devices;
        LayoutInflater inflater;

        public DeviceAdapter(Context context, List<BluetoothDevice> devices) {
            this.context = context;
            inflater = LayoutInflater.from(context);
            this.devices = devices;
        }

        @Override
        public int getCount() {
            return devices.size();
        }

        @Override
        public Object getItem(int position) {
            return devices.get(position);
        }

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

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            ViewGroup vg;

            if (convertView != null) {
                vg = (ViewGroup) convertView;
            } else {
                vg = (ViewGroup) inflater.inflate(R.layout.device_element, null);
            }

            BluetoothDevice device = devices.get(position);
            final TextView tvadd = ((TextView) vg.findViewById(R.id.address));
            final TextView tvname = ((TextView) vg.findViewById(R.id.name));
            final TextView tvpaired = (TextView) vg.findViewById(R.id.paired);
            final TextView tvrssi = (TextView) vg.findViewById(R.id.rssi);
```



```
            tvrssi.setVisibility(View.VISIBLE);
            byte rssival = (byte) devRssiValues.get(device.getAddress()).intValue();
            if (rssival != 0) {
                tvrssi.setText("Rssi = " + String.valueOf(rssival));
            }

            tvname.setText(device.getName());
            tvadd.setText(device.getAddress());
            if (device.getBondState() == BluetoothDevice.BOND_BONDED) {
                Log.i(TAG, "device::"+device.getName());
                tvname.setTextColor(Color.WHITE);
                tvadd.setTextColor(Color.WHITE);
                tvpaired.setTextColor(Color.GRAY);
                tvpaired.setVisibility(View.VISIBLE);
                tvpaired.setText(R.string.paired);
                tvrssi.setVisibility(View.VISIBLE);
                tvrssi.setTextColor(Color.WHITE);

            } else {
                tvname.setTextColor(Color.WHITE);
                tvadd.setTextColor(Color.WHITE);
                tvpaired.setVisibility(View.GONE);
                tvrssi.setVisibility(View.VISIBLE);
                tvrssi.setTextColor(Color.WHITE);
            }
            return vg;
        }
    }
    private void showMessage(String msg) {
        Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
    }
}
```

Additional XML files displayed Android layouts are available upon request.

Embedded Microcontroller System



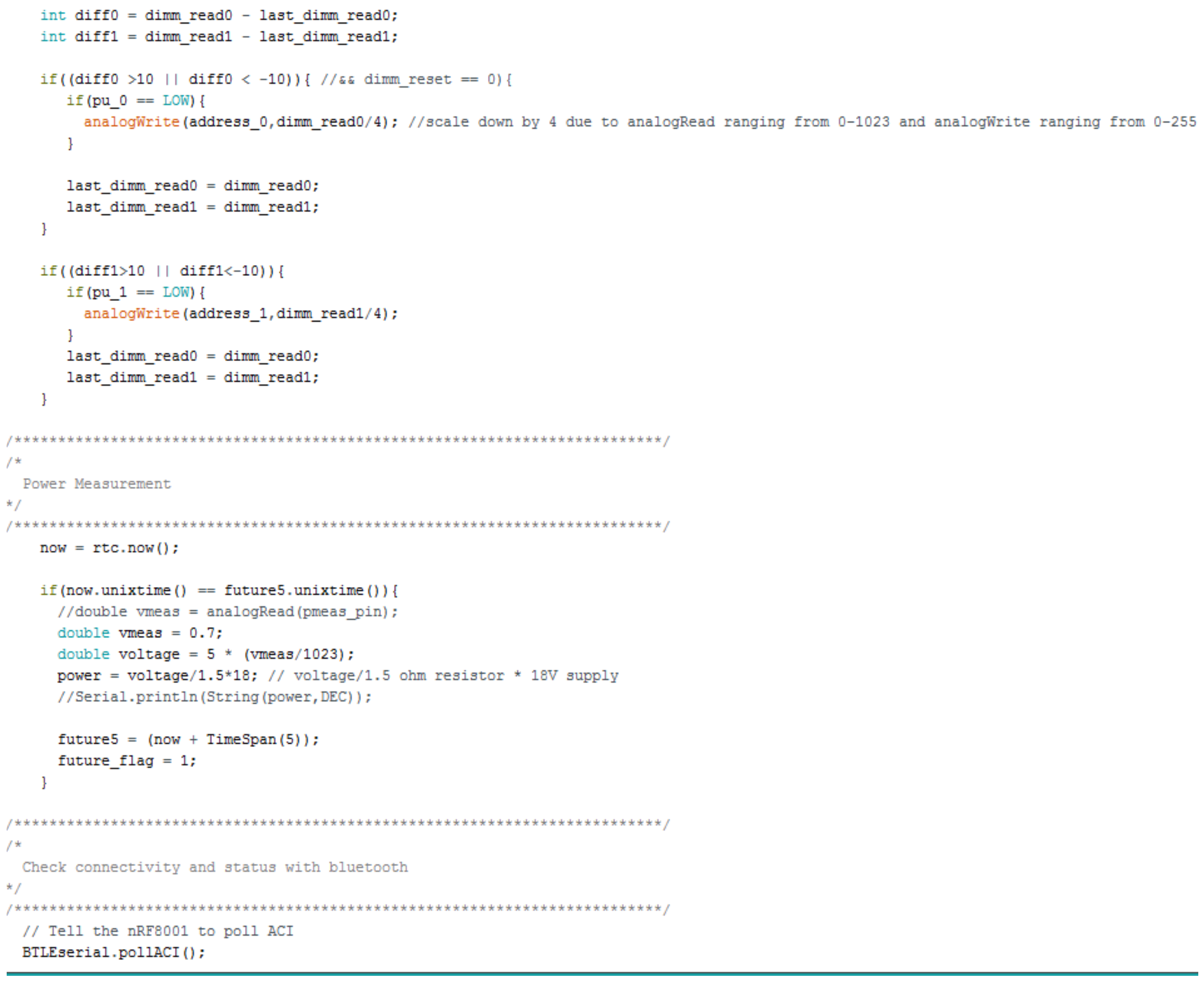
sdgt_final

```
/*********************************************************************
Senior Design - LED Lighting w/ Bluetooth Communication and RTC

By: Ryan Marion, Mitchell Wheaton

*********************************************************************/

//Libraries for communication with the BTLE
#include <SPI.h>
#include "Adafruit_BLE_UART.h"
// Date and time functions using a DS1307 RTC connected via I2C and Wire lib
#include <Wire.h>
#include "RTClib.h"

/**************************************************************************/
/*
    ATMega pin setup
*/
/**************************************************************************/
// Connect CLK/MISO/MOSI to hardware SPI
// Duemilanove/UNO - 10(SS) 11(MOSI) 12(MISO) 13(SCK)
#define ADAFRUITBLE_REQ 10
#define ADAFRUITBLE_RDY 2     // Interrupt pin
#define ADAFRUITBLE_RST 7

//Addresses for leds
int address_0 = 9;
int address_1 = 6;

//power measurement pin
  int pmeas_pin = A3;

/**************************************************************************/
/*
    Global Variables
*/
/**************************************************************************/
//Stored pwm values
int pwm[2] = {255, 255};

String out, pout;
int future_flag = 0;
```

```
String out, pout;
int future_flag = 0;

//Stored switch states
int state[2] = {HIGH, HIGH};

//Hardware switch pins and flag
int sw0 = 0;
int sw1 = 1;
int pu_0 = HIGH;
int pu_1 = HIGH;
int last_pu_0 = LOW;
int last_pu_1 = LOW;
int hw_flag = 0;

//dimmer pins and vars
int dimm0 = A2;
int dimm1 = A3;
int dimm_read0 = 0;
int dimm_read1 = 0;
int last_dimm_read0 = 0;
int last_dimm_read1 = 0;
int dimm_reset=1;

DateTime now;
DateTime future5;

double power = 0;

//Create an object for serial communication
Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY, ADAFRUITBLE_RST);

#if defined(ARDUINO_ARCH_SAMD)
// for Zero, output on USB Serial console, remove line below if using programming port to program the Zero!
   #define Serial SerialUSB
#endif

//Create an object for the RTC
RTC_DS1307 rtc;

//An array to hold the days of the week
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
```

```
/*************************************************************************/
/*
    Initialize the Arduino for serial communication and RTC communication
*/
/*************************************************************************/
void setup(void)
{
  //set up LED pins
  pinMode(address_0, OUTPUT);
  pinMode(address_1, OUTPUT);

  //set up hw switch pins
  pinMode(sw0, INPUT_PULLUP);
  pinMode(sw1, INPUT_PULLUP);

  Serial.begin(9600);
  while(!Serial);

  //Check for the RTC
  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }

  //Check to see if the RTC is running
  if (! rtc.isrunning()) {
    Serial.println("RTC is NOT running!");
    // following line sets the RTC to the date & time this sketch was compiled
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
    // This line sets the RTC with an explicit date & time, for example to set
    // January 21, 2014 at 3am you would call:
    // rtc.adjust(DateTime(2014, 1, 21, 3, 0, 0));
  }

  //Set the name we're adveritising - Senior Design Gary Tuttle
  BTLEserial.setDeviceName("SD_GT1"); /* 7 characters max! */
  //begin serial communication with the module
  BTLEserial.begin();

  now = rtc.now();
  future5 = (now + TimeSpan(5));
}
```

```
  //Checks for new events on the nRF8001
  aci_evt_opcode_t laststatus = ACI_EVT_DISCONNECTED;

void loop()
{
/*****************************************************************************/
/*
  Interface with hardware switches and dimmers
*/
/*****************************************************************************/
    //read switch states
    delay(10);
    pu_0 = digitalRead(sw0);
    pu_1 = digitalRead(sw1);
    //read dimming states
    delay(100);
    dimm_read0 = analogRead(dimm0);
    dimm_read1 = analogRead(dimm1);

    //Read the pull up resistor of the first switch
    if(pu_0 != last_pu_0){
      if(pu_0 == LOW){ //switch is closed
        digitalWrite(address_0,HIGH);
        //  analogWrite(address_0,pwm[0]);
      }

      if(pu_0 == HIGH){ //switch is open
        digitalWrite(address_0,LOW);
      }
      last_pu_0 = pu_0;
      last_pu_1 = pu_1;
    }

    //Read the pull up resistor of the second switch
    if(pu_1 != last_pu_1){
      if(pu_1 == LOW){ //if switch is closed
        digitalWrite(address_1,HIGH);
        //analogWrite(address_1,pwm[1]);
      }

      if(pu_1 == HIGH){ //if switch is open
        digitalWrite(address_1,LOW);
      }
      last_pu_0 = pu_0;
      last_pu_1 = pu_1;
```



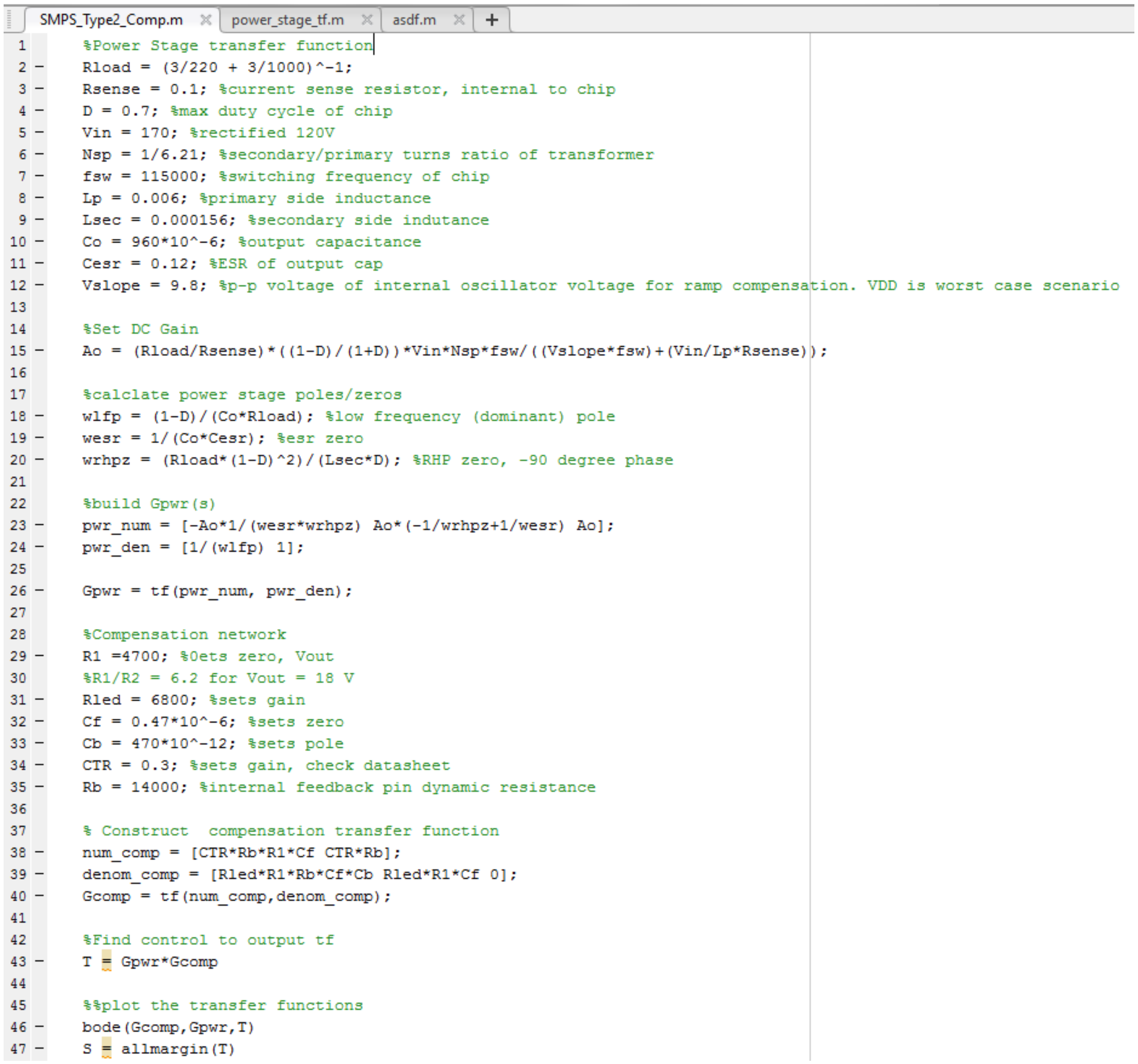
```
    int diff0 = dimm_read0 - last_dimm_read0;
    int diff1 = dimm_read1 - last_dimm_read1;

    if((diff0 >10 || diff0 < -10)){ //&& dimm_reset == 0){
       if(pu_0 == LOW){
         analogWrite(address_0,dimm_read0/4); //scale down by 4 due to analogRead ranging from 0-1023 and analogWrite ranging from 0-255
       }

       last_dimm_read0 = dimm_read0;
       last_dimm_read1 = dimm_read1;
    }

    if((diff1>10 || diff1<-10)){
       if(pu_1 == LOW){
         analogWrite(address_1,dimm_read1/4);
       }
       last_dimm_read0 = dimm_read0;
       last_dimm_read1 = dimm_read1;
    }

/**************************************************************************/
/*
  Power Measurement
*/
/**************************************************************************/
    now = rtc.now();

    if(now.unixtime() == future5.unixtime()){
      //double vmeas = analogRead(pmeas_pin);
      double vmeas = 0.7;
      double voltage = 5 * (vmeas/1023);
      power = voltage/1.5*18; // voltage/1.5 ohm resistor * 18V supply
      //Serial.println(String(power,DEC));

      future5 = (now + TimeSpan(5));
      future_flag = 1;
    }

/**************************************************************************/
/*
  Check connectivity and status with bluetooth
*/
/**************************************************************************/
  // Tell the nRF8001 to poll ACI
  BTLEserial.pollACI();
```

```
// Check current status
aci_evt_opcode_t status = BTLEserial.getState();

// detect change in status
if (status != laststatus) {
  // print new status
  if (status == ACI_EVT_DEVICE_STARTED) {
      Serial.println(F("Advertising started"));
  }
  if (status == ACI_EVT_CONNECTED) {
      Serial.println(F("Connected"));
  }
  if (status == ACI_EVT_DISCONNECTED) {
      Serial.println(F("Disconnected or advertising timed out"));
  }
  // set last status to current
  laststatus = status;
}

//Check for connection before taking changes from the phone
if (status == ACI_EVT_CONNECTED) {
  // Check for data
  //if (BTLEserial.available()) {
  //  Serial.print("* "); Serial.print(BTLEserial.available()); Serial.println(F(" bytes available from BTLE"));
  //}
  if(future_flag){
    pout = String(power, 4) + "W";
    Serial.println(pout);
    //convert line to bytes
    uint8_t sendbuffer[20];
    pout.getBytes(sendbuffer, 20);
    char sendbuffersize = min(20, pout.length());

    BTLEserial.write(sendbuffer, sendbuffersize);

    future_flag = 0;
  }
```

```
 /****************************************************************************/
/*
    Check for something to read, get a character and print it out
*/
/****************************************************************************/

    while (BTLEserial.available()) {
      char c;
      String s;
      int address;
      int state_temp;
      int ledPin;
      int buffersize = 3;
      int i=0;
      int pwm_temp;


      address = BTLEserial.read();
      state_temp = BTLEserial.read();

      for(i=0;i < buffersize;i++){
        c = BTLEserial.read();
        s = s + c;
      }

      Serial.println(address);
      Serial.println(state_temp);
      Serial.println(s);

      pwm_temp = s.toInt();
      //pwm_temp = 255 - pwm_temp; //this is when using the PWM feedback circuit

      /*************************************/
      //Address LED 0
      /*************************************/
        if(address == '0'){
        ledPin = address_0;

        //READ ON/OFF STATE
        if(state_temp == '0'){
            digitalWrite(ledPin,LOW);
            //Serial.println("OFF");
        }
        else if(state_temp == '1'){
            analogWrite(ledPin,pwm_temp);
        delay(10);

        //save pwm value
        pwm[0] = pwm_temp;
        dimm_reset = 1;
    }
    //save switch state
    state[0] = state_temp;
  }

  /*************************************/
  //Address LED 1
  /************************************/
  else if(address == '1'){
    ledPin = address_1;

    //READ ON/OFF STATE
    if(state_temp == '0'){
        digitalWrite(ledPin,LOW);
    }
    else if(state_temp == '1'){
        analogWrite(ledPin,pwm_temp);
        delay(10);

        //save pwm value
        pwm[1] = pwm_temp;
        dimm_reset = 1;
        hw_flag = 0;
    }
    //save switch state
    state[1] = state_temp;
  }

  /*************************************/
  //Send information back to the user
  /*************************************/
  out = "Light Changed";
  //convert line to bytes
  uint8_t sendbuffer[20];
  out.getBytes(sendbuffer, 20);
  char sendbuffersize = min(20, out.length());
```

```
      BTLEserial.write(sendbuffer, sendbuffersize);

    }
  }
}


// Check for data from the Serial console -- would go in the main void loop at the bottom
    /*
    if (Serial.available()) {
      // Read a line from Serial
      Serial.setTimeout(100); // 100 millisecond timeout
      String out = Serial.readString();

      // We need to convert the line to bytes
      uint8_t sendbuffer[20];
      s.getBytes(sendbuffer, 20);
      char sendbuffersize = min(20, s.length());

      //Serial.print(F("\n* Sending -> \"")); Serial.print((char *)sendbuffer); Serial.println("\"");

      // write the data
      BTLEserial.write(sendbuffer, sendbuffersize);
    }*/
```

MATLAB for SMPS Feedback Network



```
%Power Stage transfer function
Rload = (3/220 + 3/1000)^-1;
Rsense = 0.1; %current sense resistor, internal to chip
D = 0.7; %max duty cycle of chip
Vin = 170; %rectified 120V
Nsp = 1/6.21; %secondary/primary turns ratio of transformer
fsw = 115000; %switching frequency of chip
Lp = 0.006; %primary side inductance
Lsec = 0.000156; %secondary side indutance
Co = 960*10^-6; %output capacitance
Cesr = 0.12; %ESR of output cap
Vslope = 9.8; %p-p voltage of internal oscillator voltage for ramp compensation. VDD is worst case scenario

%Set DC Gain
Ao = (Rload/Rsense)*((1-D)/(1+D))*Vin*Nsp*fsw/((Vslope*fsw)+(Vin/Lp*Rsense));

%calclate power stage poles/zeros
wlfp = (1-D)/(Co*Rload); %low frequency (dominant) pole
wesr = 1/(Co*Cesr); %esr zero
wrhpz = (Rload*(1-D)^2)/(Lsec*D); %RHP zero, -90 degree phase

%build Gpwr(s)
pwr_num = [-Ao*1/(wesr*wrhpz) Ao*(-1/wrhpz+1/wesr) Ao];
pwr_den = [1/(wlfp) 1];

Gpwr = tf(pwr_num, pwr_den);

%Compensation network
R1 =4700; %0ets zero, Vout
%R1/R2 = 6.2 for Vout = 18 V
Rled = 6800; %sets gain
Cf = 0.47*10^-6; %sets zero
Cb = 470*10^-12; %sets pole
CTR = 0.3; %sets gain, check datasheet
Rb = 14000; %internal feedback pin dynamic resistance

% Construct  compensation transfer function
num_comp = [CTR*Rb*R1*Cf CTR*Rb];
denom_comp = [Rled*R1*Rb*Cf*Cb Rled*R1*Cf 0];
Gcomp = tf(num_comp,denom_comp);

%Find control to output tf
T = Gpwr*Gcomp

%%plot the transfer functions
bode(Gcomp,Gpwr,T)
S = allmargin(T)
```