**SmartBike**

Michael (Jake) Goulding

Stephanie Howorth

Suraj Mistry

Syed Nabi

Eric Phillips-Sheldon

**Abstract:**

The concept of a human-powered vehicle has peaked the interest of engineers dating back to the 1400s. It was not until the later 1800s that the concept of a bicycle was realized for transportation. In recent decades, bicycles have become a second class form of transportation, used primarily for the purposes of recreation and health. Utilizing bicycles for the purpose of clean transportation is not as commonly recognized as an incentive to ride. The SmartBike project presents a system in which users are able to apply an entertaining and innovative spin on their old bicycles. It provides a recreational incentive for users to maintain an active, safe and environmentally sustainable lifestyle. The SmartBike system is a smart retrofit kit consisting of an energy-harvesting circuit, Arduino microcontroller and user-friendly application. These three components come together in a fully functional product to alter the way people perceive the functionality of bicycles.

**Introduction**

The goal of this project was the creation of a kit that could be easily added to any bike, that would help make biking more accessible and enticing to the average commuter or resident. Currently, most smart bikes are built into an existing bike, and are mostly marketed towards high end heavy bike users, not the casual user.

This kit can be added to any bike with relative ease, and adds a number of useful features. The turning lights are a much more familiar concept and device than the hand signals traditionally used in bikes, and are more recognizable by cars as well. The power generation is a very sustainable method of keeping the microcontroller on the bike charged, as well as charging a connected Android phone. The phone and the microcontroller can communicate through the application, allowing the user to see information about their ride. The application provides the user information such as GPS location, along with nearby restaurants, provides fitness information, and tells the user how much money they have saved on gas.

**Design Methodology:**

**Application Interface – Design**

Please view Part I of the Appendix in this document for the application design and a brief background regarding the Android platform.

**Application Interface – Assumptions, Errors and Solutions**

Several assumptions were made when proposing the design for the application’s interface and logic. These assumptions were made without any knowledge of the software structure for the Android platform. One assumption made involved data transfer within the application. It was anticipated basic global values could be established within one class and passed to the others using imports. However, despite being java classes, the Android platform cannot perform this task with ‘Android Activity’ classes. Described below are two methods utilized in an attempt to establish global variables within the application.

**Method One – Xml File**

All resources used within the Android platform are hardcoded and stored in a xml files located in a resource directory following the path *SmartBike/res/values*. Following this principle, the idea was to create an xml file containing a data type pertaining to the user’s profile and Bluetooth information. The intention was to create a data type which can be accessed from one java file to receive and/or manipulate the data and a separate java file to display the extracted data to the user.

The issue with this implementation is that any values identified in an xml file are statically defined. They cannot be manipulated during run time. They cannot be created or manipulated by the application and have to be done manually by the programmer. This implementation is sufficient as a method to obtain Bluetooth information, provided the microcontroller sends xml files in the proper format to the application. However, it does not enable the user’s input to be stored and used for calculations.

**Method Two – Shared Preferences**

Within the Android platform, SharedPreferences is a class which enables a user to hardcode new information into the application based on their input. This gives an Android application the ability to create password protected profiles. The issue with this implementation is it is designed to save protected information. Even if the protected values are cast into other variables and/or utilized in calculations, they will not show on the user’s screen.

**Solution**

Although a hardcoded establishment of global variables is ideal, the Android platform has a bit of a restricted environment in this respect. In order to pass global variables throughout the application, a separate java class had to be created. The class consists of various sets and gets for all global variables, as well as performs all calculations required within the application.

**Microcontroller: Arduino UNO**

The microcontroller unit acts as one of the processing units for the system. It controls the turning lights, counts the revolutions made by the wheel and communicates to PC or Android applications via Bluetooth. It is powered by the energy harvesting circuit.  
  
**Turning Lights:**

Initial design: Initially the turning lights were implemented using polling where the loop constantly polled the state of the push buttons (left and right) to check for user input.  
  
Improved design: The former design was then improved by implementing interrupt service routines (ISR). In this current design, each push button is attached to an ISR that toggles a flag. The flag then triggers a blinking method for the LEDs. So, when a push button is pressed, the respective light starts blinking and turns off at a second push of the button. Each LED has a 47kΩ resistor in series with it and is powered by the 5V pin from the microcontroller. The same circuitry and code was implemented for both turning lights.

**Revolution Count:**

The hall-effect sensor is placed in close proximity to the spokes of the back wheel. It is powered by the 5V source drawn from the microcontroller with a resistor of 47kΩ in parallel to it. A magnet is attached to one of the spokes and an interrupt service routine is implemented to detect the crossing of the two. Every time the magnet crosses the hall-effect sensor, an interrupt is triggered and the service routine increments a counter.  
  
**Bluetooth:**

The communication to PC or Android application is maintained via a Bluetooth connection by the Arduino. A Bluetooth module is attached as a peripheral to the board. This peripheral will send any serial data printed out by the Arduino code out over the Bluetooth connection. The code used waits for a request from the connected device and sends the elapsed counts i.e the number of revolutions made by the rear tire since the last request and time elapsed since last request. Data is only sent when a request is made. Every time the Bluetooth receives a ‘1’, it sends the data out over the Bluetooth connection.

**Analysis:**

**Application Interface**

In early stages of the design process, an emulator was utilized to simulate application functionality. A mobile device emulator is incorporated within the Android SDK package. It enables programmers to debug without physically connecting to a mobile device. This process was scrapped early in the design process as it is rather ineffective and impractical. The emulator itself is extremely slow to launch and does not provide external features such as Bluetooth or GPS.

Most of the debugging process was made by physically downloading the application onto a mobile device using a USB connection. If the need for trouble shooting were to occur during trials, break points were set in place to isolate the issue.

**Implementation:**

**Application Interface -Calculations**

Calorie Counter:

An assumption made initially was to establish calorie calculations based on slope, acceleration and various other variables. Details regarding the proposed calculation are displayed in Figure 2 of the Appendix. The simplified calculation can be found in the same reference used for the proposed calculation. [7] It eliminates the portion of the equation containing acceleration and slope. They are variables which account for minor discrepancies with calculation error.

**W = Speed (K1 + K2\*Speed\*Speed) [joules/sec]**

where K1 and K2 are constants based on bike weight

(please see Appendix figure 2 for constant details)

**Calories = W\*time / (4186 joules per calorie)**

Gas Saving Counter:

**Money saved = Gas Consumption (L/km) \* Distance**

**Travelled (km) \* Cost of Gas ($/L)**

Determining Speed Based on Revolutions:

**Speed = 2 \* π \* (radius bike wheel) \* revs [m/s]**

**Application Interface - Proposal Deviations**

The original design was created with the intention to perform tasks that exceeded the Bronze level deliverables. Specifically, the usage of social media was not integrated into the final prototype. The original user profile was designed to have the user sign in to their social media. This login feature would enable the application to post milestones the user would encounter during their rides. Unfortunately, this implementation exceeded the scope of the project. In addition, implementing a lock feature and a social racing feature also exceeded the project’s scope.

**Application Interface – Implementation**

Please view Part II of the Appendix in this document for figures, code and full descriptions of the application’s design and framework.

**Analysis:**

**Bluetooth Theory**

The Android application was determined to have many functionalities but among the skeleton of it included Bluetooth data transfer that was an integral addition to the phone’s contribution to this project. Group H12 had analyzed different available method of implementing data transfer between the Arduino and Application. Serial transfer through Micro USB was avoided because of difficulty, Android does not allow for easy data transfer through applications with that means. WiFi was avoided because then the user would be inclined to maintain a hotspot to connect the Arduino to then have that running throughout their entire bike ride which was inefficient use of resources because Bluetooth implementation was available. Initially the application wasn’t going to require the phone to be paired to the device, and it would display a list of devices for the user to tap and connect from. Before this could be implemented a fundamental learning of Android programming was required.

**Method Three – Bluetooth Connection**

The first step was to install the required applications and tools to have a proper “Hello World” application emulator run in a simulation window.  Eclipse, was downloaded as well as SDK manager to install all the required libraries and a large problem that was encountered was finding the “ADB Device Manager” because the file organization of the Eclipse IDE as an application once it was downloaded did not maintain integrity according to online posts.  I was required to launch Windows Powershell and execute a command to stop the running ADB manager such that Eclipse could find the proper extensions and tools to run a basic “Hello World” application.

Next was to begin running a virtual simulator of a device and launch a running “Hello World” application on the simulated device, this did not prove to be too difficult except for creating the appropriate parameters of the device to simulate an accurate representation.  Once the simulator worked and loaded the app, then began the creating of buttons and menus to navigate from page to page.  This proved to be a challenge without proper direction, so a textbook on Android Application Programming was purchased and significantly lightened up the difficulty in the process.  Once a page intent was created and a base understanding was established, Steph continued on working on the General User Interface and displaying information stored by the app in its respective page, and Suraj began research on Bluetooth connectivity.

**Simulating**

First step Suraj took in constructing a successful Bluetooth connection with the micro controller was searching for similar examples of Android projects to look for what was required.  Initially when working on separate project, the issue arose that there was no Bluetooth functionality of the desktop that was responsible for that component of the project.  Once an adapter was bought, around Christmas the computer to have the adapter installed on corrupted so all progress needed to start from scratch with reinstalling Eclipse. The problem with majority of these projects was that they all implemented Bluetooth connections to other Bluetooth devices.  A particularly helpful website [13] was found to understand the basics of what happens when one device communicated with another device.  It was understood that once a socket was created via Bluetooth, and had expired its use, the running thread needed to be closed to save on resources on the device.

Once a successful simulation of Bluetooth code was sent to the virtual simulation another problem was encountered.  The computer at use didn’t have any Bluetooth adapter to emulate the simulation having Bluetooth capabilities, so testing on the phone began.  An application to turn on and off the Bluetooth module on the phone was created but a “PERMISSIONS” error was encountered.  It was discovered the Manifest.xml file needed to be modified to implement the required Bluetooth operations such that the application had the appropriate administration over the right Bluetooth module of the phone.

Around here was where the computer that was running the Bluetooth application began to slow down drastically and eventually would stop running so precautions were taken to prevent the computer from damaging itself and Windows needed to be reinstalled as well as the entire project.  This problem was solved by wiping the harddrive and reinstalling Windows, followed by having to go through the previous steps a second time without most of the required troubleshooting.

**Master-Slave Research**

It was discovered that there is a Master-Slave implementation to the Bluetooth operations in the application.  One side of an application needed to create a “BluetoothServerSocket” and research was done based on the Android library. [4]   Another difficulty encountered was realizing the Arduino Uno’s Bluetooth module was unable to connect to a BluetoothServerSocket manually and the Android Application needed to do more than simply create a ServerSocket.

**Socket Research**

To overcome the knowledge gap of sockets and Bluetooth implementation procedures, Android Studios offered helpful libraries and help guides. [3]  A difference between referencing “createRfcommSocket” and “createRfcommSocketToServiceRecord” was discovered to create the appropriate sockets once the device attempting to connect to was retrieved.  Another problem encountered was that the Arduino could not be paired with because in multiple example codes the procedure would search for a device as it appeared as an “intent” or as the Bluetooth module on the phone discovered the Arduino while the device was already paired with it.  This caused an error of the phone never actually seeing the Arduino as a discovered “intent” so another method was necessary.

**Discovering Paired Devices**

At this moment a lot of adversary because an example of connecting to a paired Arduino was unavailable, as well as it was assumed the Arduino had a specific UUID code to pair with.  The UUID stands for  Universal Unique Identifier and it was discovered that the Arduino doesn’t require this method of pairing, which resulted in a standardized UUID being implemented.  Once this was establish  and finding the device once already paired to it presented a difficulty, Suraj discovered a way of finding the device name in a library of BluetoothAdapter().  He then found the devices the phone was currently paired to using getBondedDevices() and created a socket to write to and from based on that device’s name

It was assumed that since the Arduino was acquired by the code, and a dedicated socket was created sharing the standard UUID with the same Arduino, yet it still would be unable to create an active socket, so further research was conducted.  Initially, it was determined that there needed to be a running Thread for the BluetoothSocket() class to create a socket successfully and eventually terminate it, as stated in an example discovered online. [2]

Following that, a InputStream and OutputStream was created to send and receive data accordingly to the BluetoothSocket using examples of data transfer. [5]

After most of the code was put together using different examples and implementing code changes based on errors thrown using breakpoints it was realized that there was a dedicated Library for the Bluetooth Module attached to the Arduino and tcode was analyzed for further information. [10]

**Success!**

Finally, the socket was still struggling to be implemented most of the time and the error was discovered to be that the “createRfcommSockettoServiceRecord()” only accepts UUID with “mPort” parameters set to 1, and in  Android 4.2 and greater this value is automatically set to be -1, so in figure BaseActivity.java line 84 needed to be implemented to consistently create a steady connection, and this was discovered in an online example as well. [9]

**Microcontroller Implementation**

**Parts**  
Microcontroller : Arduino UNO  
Hall Effect Sensor: IC Sensor Hall Effect 3SIP  
Magnets:  
Push Buttons: Switch Tactile SPST-NO 0 05A 24V  
LEDs : LED Red Diff 5MM Oval T/H  
Bluetooth : Sunfounder HC-06  
  
**Micrcontroller : Arduino UNO**

Initially, while deciding which microcontroller to use for the microcontroller component of the project, various requirements were considered such as price, power consumption, and suitability for the projects needs. Some of the initial requirements were for wireless connectivity and for storage. For wireless connectivity, WiFi and Bluetooth were examined. For storage, SD or microSD technology were examined. After an analysis of Bluetooth and Wifi, Bluetooth was decided upon due to lower power draw and simplicity in connection with Android applications. MicroSD proved to be not necessary considering the quantity of data that needed to be stored.  
  
**Hall Effect Sensor: IC Sensor Hall Effect**  
In deciding upon which hall effect sensor to use, the size of the sensor itself was a vital point to consider since it needed to fit onto the frame of the bike right beside the wheel. It was important that the sensor was small enough in size to place without interfering with the operation of the bike. Although soldering the sensor pins onto wires proved to be a challenge initially due to the heat produced by the iron, it was later on installed using a header socket to prevent heat issues.

**Magnets***:*  
The magnets were decided based on the strength. Four ¼” inch magnets were attached onto a spoke of the wheel. It had a magnetic range of approximately a centimeter.   
  
**Push Buttons:**

Two simple one pole push buttons were used in order to take inputs for navigation from the user. A 3.9kΩ resistor and a 10nF capacitor were used to implement debouncing for each of the push buttons and a 5V line was drawn from the microcontroller’s 5V pin to power the buttons.   
  
**LEDs:**Since the target was to make a prototype of the device, a simple LED was chosen for this project. These LEDs drew power from the microcontroller in series with 47kΩ resistors and were grounded on the bike frame.  
  
**Bluetooth : Sunfounder HC-06**

The Bluetooth module was decided based on the popularity and availability in the seller’s site. Good reviews on the sellers site, as well as price point and speed in delivery, were also factors in choosing the module.  
  
**The Code:**

In order to code the functionalities of the microcontroller, different new functions were defined (as given in the source code) which were then plugged into the main loop in the code. All the interrupts were attached in the setup and all the variables and pin assingments were declared outside the setup. Also an arduino library, *PinChangeInterrupt.h,* was introduced in order to broaden the number of pins that could be used as interrupts.

**Sustainability and Social Impact:**

If the SmartBike retrofit kit was to be produced as a product, it could potentially produce several sustainable effects and have a measurable social impact. It would encourage biking both recreationally and as an alternative to driving while commuting. It would also work to improve engagement in the local community through targeted advertisements for local businesses. In addition, it would encourage longer distance biking, as the phone charging capability allows users to maintain charge to their phone throughout the ride. Additionally, it would improve bike-car interactions, as the turning signals, and other potential future improvements, would make the bike appear as a more familiar vehicle to the car driver, and make the bike user more visible and safe while biking at night.

The SmartBike features, namely accurate distance measurements, time spent biking, calorie count, and money saved on gas, can act as encouragement for both recreational and commute style biking. Recreationally, it provides the user with an accurate measurement of their current biking activites, and can keep a log of past activities, allowing the user to see improvement and be encouraged to bike more. For commuting, the user is able to see how much they are saving on gas, as well as being able to charge their phone and signal in a way recognizable to car drivers. This means that a commuter can measurably see how they are helping the environment and saving money, can keep their phone, a very important device for modern workers, fully functional, and can make commuters feel safer while having to commute on roads without designated bike lanes.

The SmartBike would also work to improve engagement in the local community through targeted advertisements for local businesses. A local business which advertised on the SmartBike app would be able to directly target cyclists in the area. Targeting an audience which is conscious about their carbon foot print and willing to put in effort towards more environmentally friendly measures would encourage the development of green, sustainable local businesses. This directly targeted communication between bikers and store owners would encourage biker engagement within the community, and make biking a more desirable method for local transportation in an area.

In addition, it would encourage longer distance biking, as the phone charging capability allows users to maintain charge to their phone throughout the ride. Long hours of operation drain the battery from phones and other devices. This can be discouraging for riders planning on going long distances, as phones and other devices are often used to enhance the biking experience, whether with music, directions, or other methods. With the SmartBike’s power generation capabilities, the devices can remain fully charged and operational throughout the ride, taking away a potential source of discouragement from long distance bike rides, which promote health, fitness, and physical well-being.

Bike-car interactions are a potentially lethal area for cyclists. According to the CAA, 7,500 cyclists are seriously injured every year, one third of cyclist deaths occur at night, and cyclists are most likely to be killed at signal controlled intersections. [6] The Ontario government passed updated bike safety laws in 2015 that increased the fine for cyclists without lights, a reflector, and reflective tape. [8] The SmartBike turning signals would act to make cyclists more visible to cars in the dark, and would help provide clarification as to a cyclists intention when in a signaled intersection. This would allow for an increase in the bike mode of transportation, as biking would be a safer and more viable option.

**Team Organization:**

The team was divided up into several groups in accordance with the part of the project the group member was to work on. The three groups were the application group, consisting of Stephanie and Suraj, the microcontroller group, consisting of Eric and Syed, and the electrical group, consisting of Jake.

Each group worked mostly alone, with occasional check ins with other groups to ensure that parts were remaining compatible. Meetings were often held following bi-weekly meetings with Dr. Hranilovic. Communication between the entire team was carried out on a continuous basis through a messenger app, which was used to collaborate on meeting times, share ideas, and ask for help.

The application group divided their work with Suraj working on the Android Bluetooth, and Stephanie working on the user interface and calculations. They shared their work via sending it back and forth over a messenger service.

The microcontroller group divided their work between the turning lights and hall effect sensor and the Bluetooth module. Syed worked on the turning lights and hall effect sensor while Eric worked on the Bluetooth. They met to work together on parts where difficulty was being had, such as the switch from polling to interrupts. Code was shared via a Github repository.

The electrical group consisted of just Jake, and so he worked alone for the most part. He was occasionally joined in the lab with other members of the team, mostly for moral support.

**Conclusion:**

**Electrical**

Out of all the components that was initially estimated to be required, the circuitry was under-estimated because of the simulation implemented. A boost chip was placed on the bread board because in the early testing phase a drill was used to rotate the bottle generator. The generator would not consistently output 5V when “pedalling”, or spinning the drill, which produced less than required amount of power for the power bank to store. This led to purchasing a boost chip to boost the output of the bottle generator attached to the wheel. Once implemented, the wheel yielded a drastically higher rpm on the generator than the drill could simulate because of the high gear ratio. In further iteration the boost chip would be removed and the bread board would be replaced with a Printed Circuit Board to reduce the amount of wire oxidation, and chip malfunction due to connection issues.

**Microcontroller**

The Arduino Uno was a huge success and out of the initial requirements set for the group, turning lights, Hall Effect sensor, as well as Bluetooth Connectivity was successfully implemented. In the next iteration of the SmartBike, added features would be head lights and brake lights. Both of these features will add to the night-friendly appeal of the turning lights and was not accomplished in this iteration due to many technical issues encountered with the Hall Effect sensor. Indicator lights would be added to the bike as well as further group work with the Application to increase the functionality of the Bluetooth connection. A gyroscope could be implemented to sense turns and send data to the application with benchmarks along routes as added functionality that missed being included in this iteration.

**Application**

The SmartBike application group H12 implemented turned out to have the visual appeal and technical prowess to tie the entire project together with the flawless aesthetic of a professional fitness device. A single profile is able to be created and the User is able to set their custom wheel diameter, bike weight and type, as well as the user’s weight, gas efficiency of their vehicle, and the price of gas to calculate all of the desirable information the user is lacking in their current endeavours when using their bike. Social media integration as well as implementing a stationary game virtual reality were not achieved because of the great deal of adversaries faced. Communicating more with the other teams would have shortened the errors and in the next iteration will be addressed such that separate teams comment code and communicate with each other outside of respective team as well as within.

Bibliography

1. *2014 fuel consumption guide*. Ottawa: Natural Resources Canada, Office of Energy Efficiency, 2014.
2. android?, How. "How To Create A Bluetooth Server Socket In Android?". *Stackoverflow.com*. N.p., 2016. Web. 11 Apr. 2016.
3. "Bluetooth: What Is The Difference Between Bluetoothsocket And TCP IP Socket? - Quora". *Quora.com*. N.p., 2016. Web. 11 Apr. 2016.
4. "Bluetoothserversocket | Android Developers". *Developer.android.com*. N.p., 2016. Web. 11 Apr. 2016.
5. C, Scott. "Arduino Basics: Bluetooth Android Processing 4". *Arduinobasics.blogspot.ca*. N.p., 2013. Web. 11 Apr. 2016.
6. Canadian Automobile Assocation, "CAA bike safety," 2011. [Online]. Available: http://bikesafety.caa.ca/cyclists/bicycle-statistics.php. Accessed: Apr. 11, 2016.
7. D. G. Wilson, J. Papadopoulos, and F. R. Whitt, *Bicycling science*. Cambridge, MA: MIT Press, 2004.
8. Government of Ontario and Ministry of Transportation, "Bicycle safety," 2015. [Online]. Available: http://www.mto.gov.on.ca/english/safety/bicycle-safety.shtml. Accessed: Apr. 11, 2016.
9. IOException: read failed, socket might closed - Bluetooth on Android 4.3. "Ioexception: Read Failed, Socket Might Closed - Bluetooth On Android 4.3". *Stackoverflow.com*. N.p., 2016. Web. 11 Apr. 2016.
10. "Johnhowe/Blueterm". *GitHub*. N.p., 2014. Web. 11 Apr. 2016.
11. L. Darcey and S. Conder, *Sams teach yourself Android application development in 24 hours*. Indianapolis, IN: Sams, 2012.
12. “Unleaded Gasoline | Esso Pricing,” *Esso*. [Online]. Available at: *http://www.esso.ca/unleaded-gasoline*. [Accessed: 10-Apr-2016].
13. Wirsing, Brian. "Sending And Receiving Data Via Bluetooth With An Android Device". *egr.msu.edu*. N.p., 2014. Web. 1 Nov. 2015.

**Appendix 1: Application**

**Part I:**

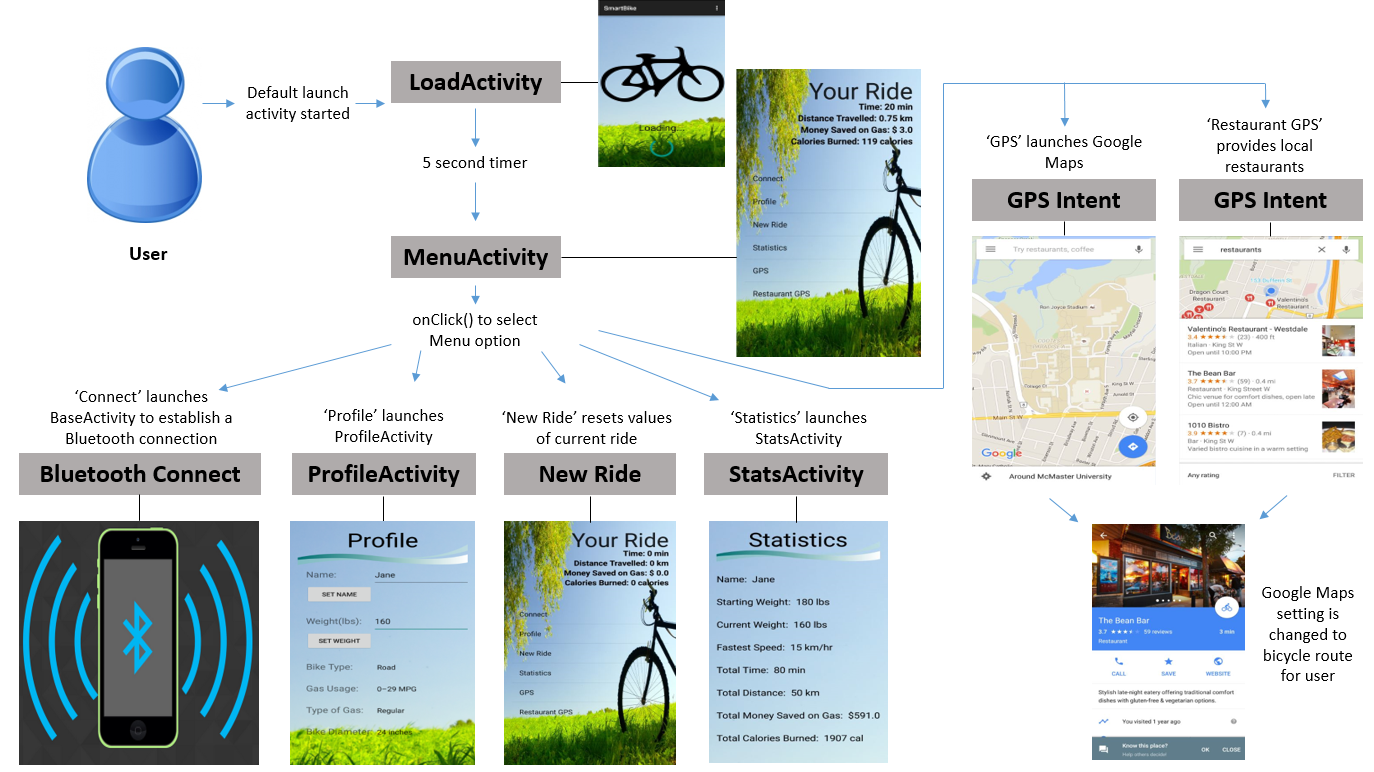
In order to properly understand the structure and framework for the application’s interface, a basic understanding of the Android platform needs to be established. The framework for the Smartbike prototype application is depicted in Figure 1.

The framework for the application can be divided into two sections, screens and intents. Screens are the visual component to the application and compose the interface the user views during run time. Each screen has a corresponding ‘Activity’ file. These ‘Activity’ files are java classes that outline the underlying logic behind the graphical interface. The graphical interface’s structure of a screen is identified with a corresponding xml file. These xml files define the text boxes, text sizes, images, lists, input boxes and various other fields for the screen’s interface.

|  |  |
| --- | --- |
|  | Xml files identify more than the graphical user interface for screens. They contain hardcoded information regarding all the Android application’s resources and values.  The resource directory categorizes the various xml files used within the application. The screen xml files described above are set within the directory Smartbike/res/layout. All hardcoded values are within the Smartbike/res/values directory and identify the strings, dimensions, Booleans, colors and styles utilized throughout the application. Other features are also identified in the resource directory. For example, the ‘anim’ folder contains separate xml files to enable a ‘fade in’ and a ‘fade out’ animation feature for the loading screen.  The most important xml file within the application is the AndroidManifest file. The nature of this file is similar to that of a ‘make’ file. It outlines the parameters to initialize the application. The manifest file identifies the target application’s name, supported Android devices, screen activities and permissions. Permissions are essential to the execution of this application. Not only does it enable the application to access Bluetooth and Google Maps but it gives the application administrative access to Bluetooth to establish a connection, create sockets and receive data. |

The second component to the framework of the application is an intent. An intent is an object within the android operating system that handles task requests. It enables the application to launch from one screen to another or pass information to and open Google Maps.

**Figure 1:**



This diagram maps the user’s experience through the SmartBike application. It gives a visual representation of how the user will navigate through different features.

**Figure 2:**

**W = Cv [K1 + {K2 (Cv)\*(Cv)} + {10.32Em (s/100 + 1.01a/g)}]**

|  |  |
| --- | --- |
| **W = Watts**  **1 Calorie = 4186 Joules**  **Cv = Speed of cyclist in m/s**  **Em = Mass of cyclist and bike**  **s = Slope or grade in %**  **a = Acceleration of bike in m/s2**  **g = Gravitational acceleration in m/s2** | K1 and K2 are constants based on the weight of the bike:  https://lh4.googleusercontent.com/a04an7eMq6jK3Zihy4GIXzie8vu-2wTsuH07K0mWJu7mK90eJWJUln3gQhQEOQpez-652cuc0uBrX8mK2wNvVxUWU9wRTXiQwAY0nLEUAuMqh51oCntvLfmeBONTI3E_-1L30HSp |

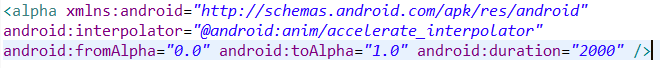
**Part II:**

**Loading Screen**

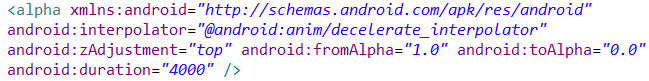
|  |  |
| --- | --- |
| The loading screen provides a 4 second timer to allow the application to initialize. When the timer has reached 4 seconds, a Handler activates the animation xml files, activityfadein.xml and splashfadeout.xml | **Load.xml** |

Fade in and fade out features indicate the duration for one screen to fade in, (Menu Screen) and which screen to fade out (Loading Screen).

**activityfadein.xml**



**splashfadeout.xml**



**LoadActivity.java**



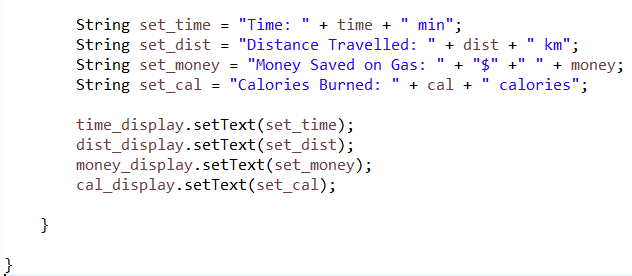
**Menu Screen**

|  |  |
| --- | --- |
| The Menu screen is the core screen for the application’s navigation. It features a list of several different menu options. An onClick() listener runs constantly in the background to determine when an option has been selected. Each selected option has its own respective task.  **Connect**   * Launches BaseActivity.java and menu\_item.xml files. * Establishes a Bluetooth connection and receives data from the microcontroller.   **Profile**   * Launches ProfileActivity.java and profile.xml files. * Obtains user input for calculations * Default options are automatically set   **New Ride**   * Sets current statistics to zero * Current statistics are displayed on the Menu screen   **Statistics**   * Launches StatsActivity.java and stats.xml files. * Displays calculations and profile information   **GPS/Restaurant GPS**   * Uses an intent to launch Google Maps * User’s location is passed when launching the new application | **Menu.xml** |

**MenuActivity.java**







**Profile Screen**

|  |  |
| --- | --- |
|  | The profile screen enables users to input their information to personalize the calculations displayed throughout the application. There are default options set when first installing the application. Any changes in information are automatically applied to all calculations.  Due to the user input feature, the profile screen is quite hefty in terms of the volume of code used. The structure of this screen consists of several buttons, input text fields and spinners. Spinners provide a list for the user to choose from. Each component and their respective options have logic applied to it within the ProfileActivity.java class. |

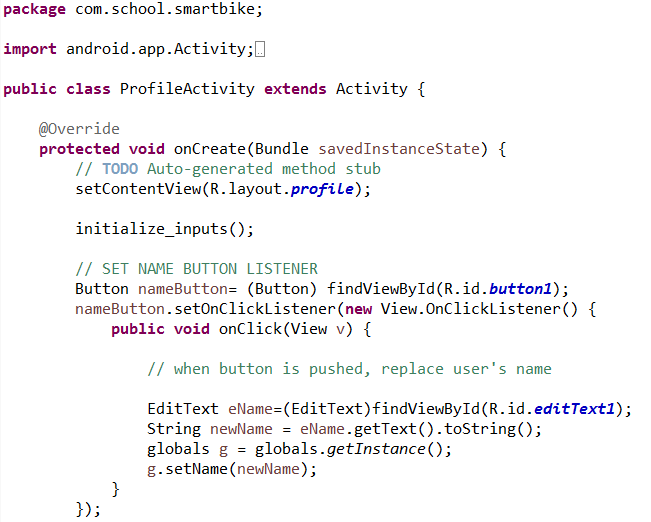
**Profile.xml** - Lines 1-84 Lines 85-165

|  |  |
| --- | --- |
|  |  |

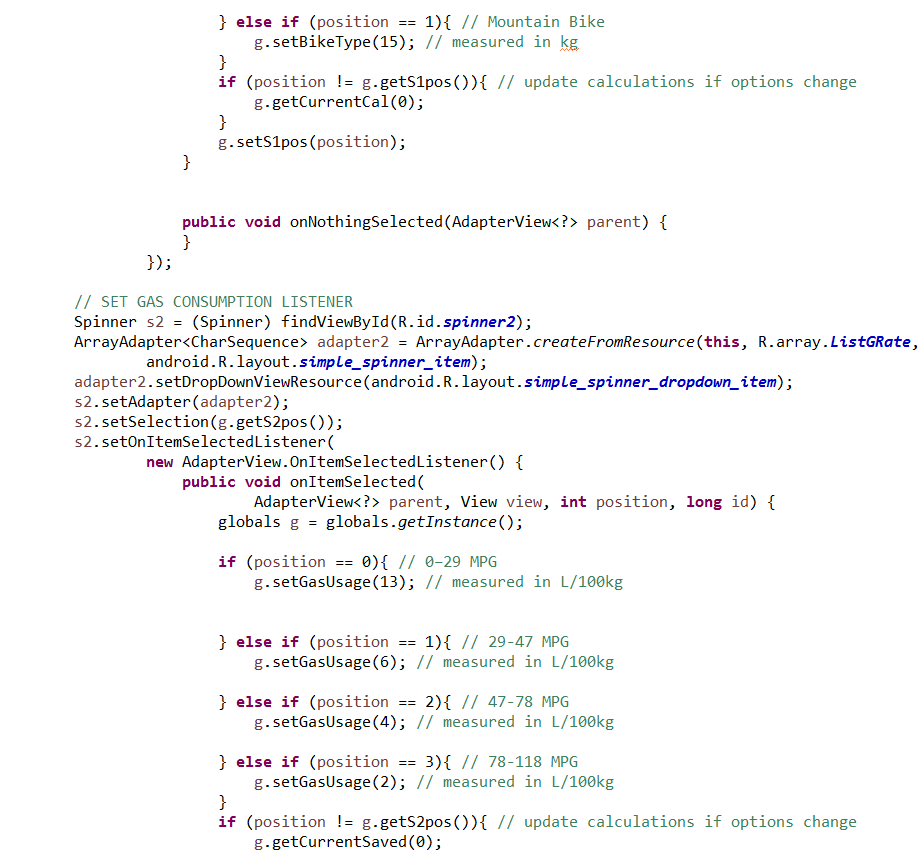
Lines 166 – 201 Lines 202 - 269

|  |  |
| --- | --- |
|  |  |

**ProfileActivity.java**

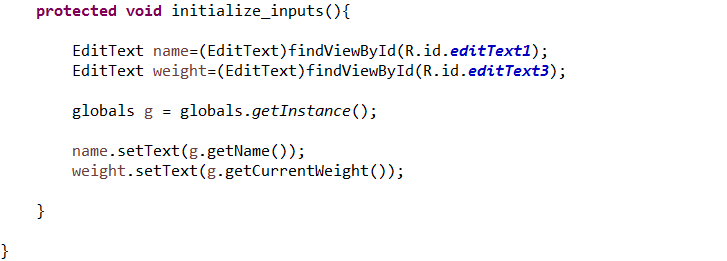












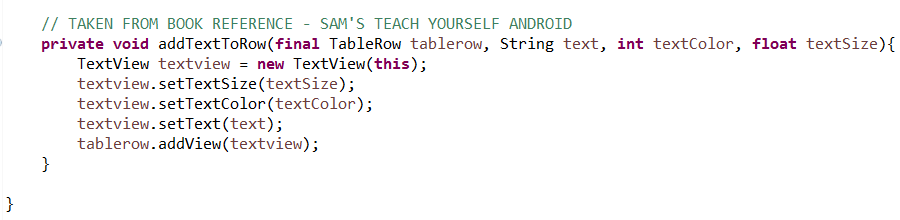
**Statistics Screen**

|  |  |
| --- | --- |
| The statistics screen contains user profile information as well as displays several calculations made throughout the application. | **Stats.xml** |

**StatsActivity.java**







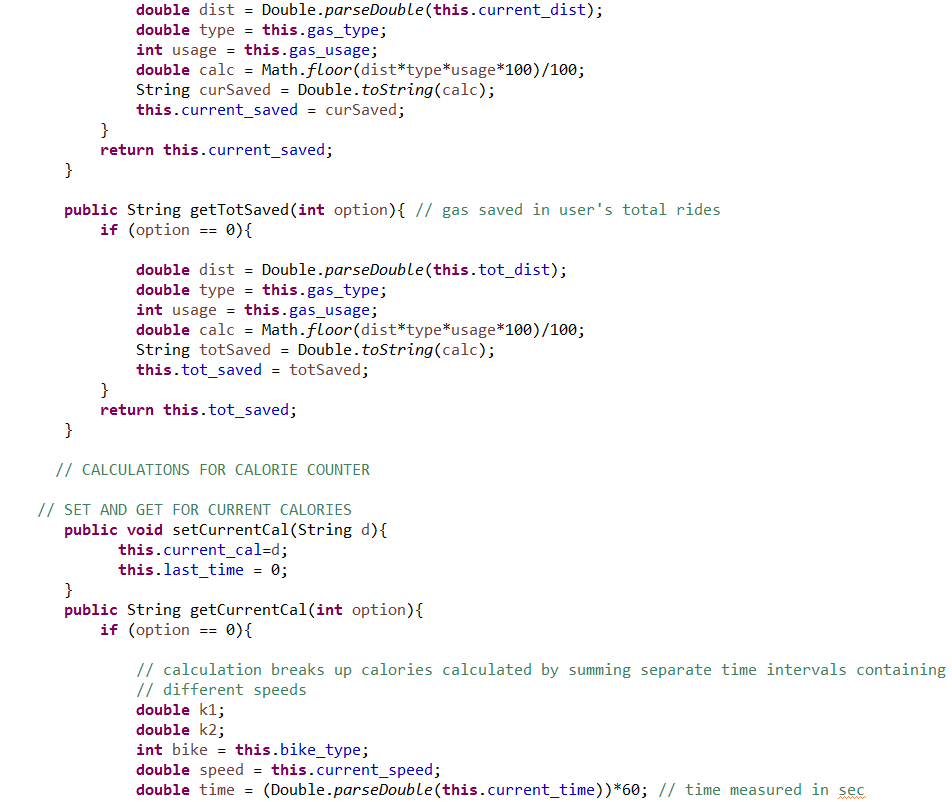
**AndroidManifest.xml** – similar in nature to a ‘make’ file

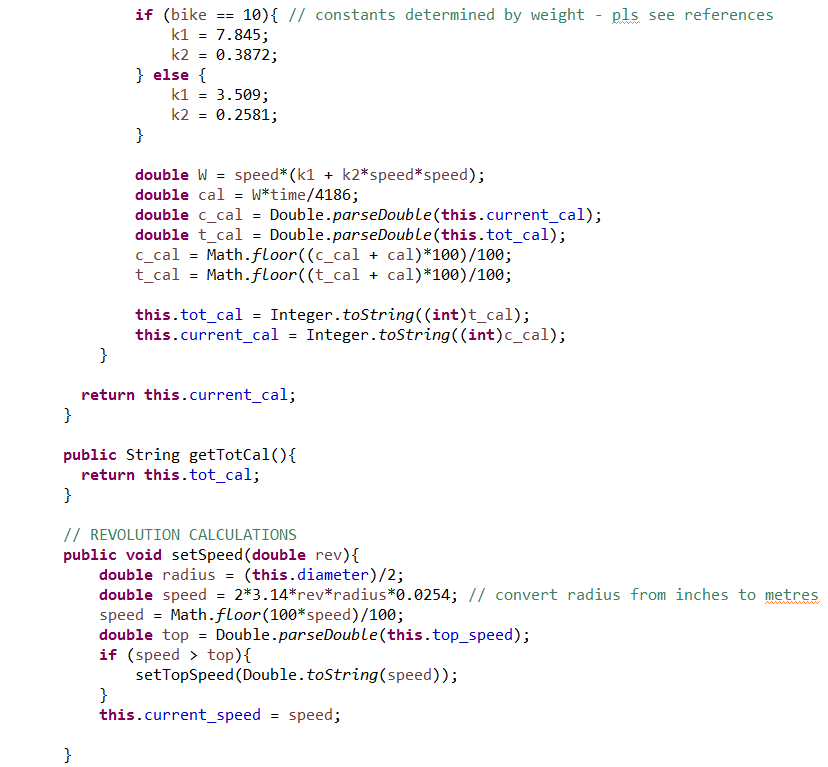
The most important xml file within the application is the AndroidManifest file. The nature of this file is similar to that of a ‘make’ file. It outlines the parameters to initialize the application. The manifest file identifies the target application’s name, supported Android devices, screen activities and permissions. Permissions are essential to the execution of this application. Not only does it enable the application to access Bluetooth and Google Maps but it gives the application administrative access to Bluetooth to establish a connection, create sockets and receive data.



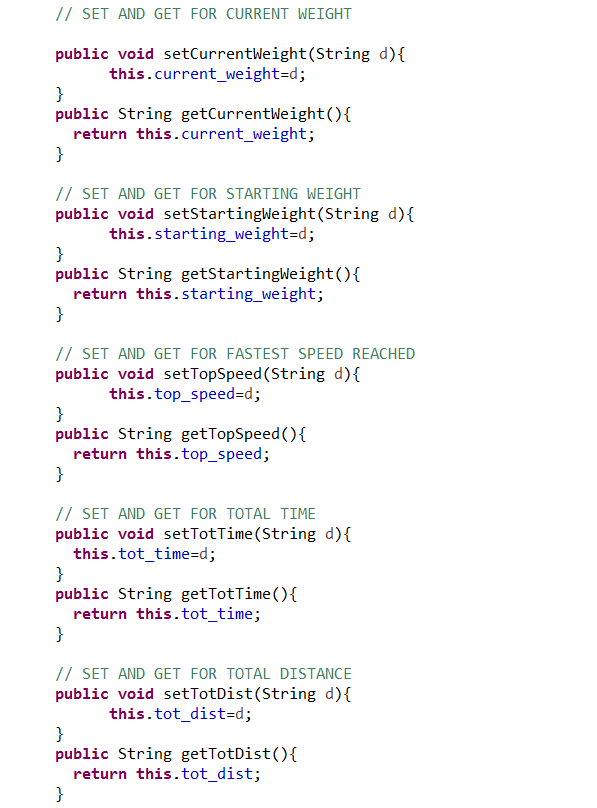
**Globals.java** – contain all global variables and calculations

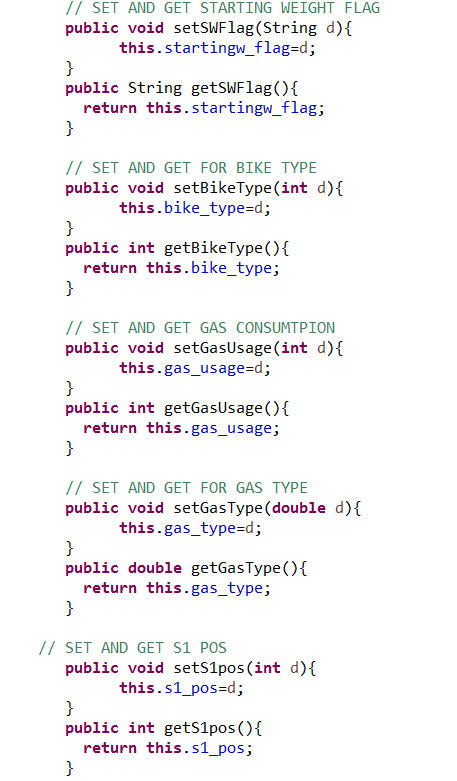


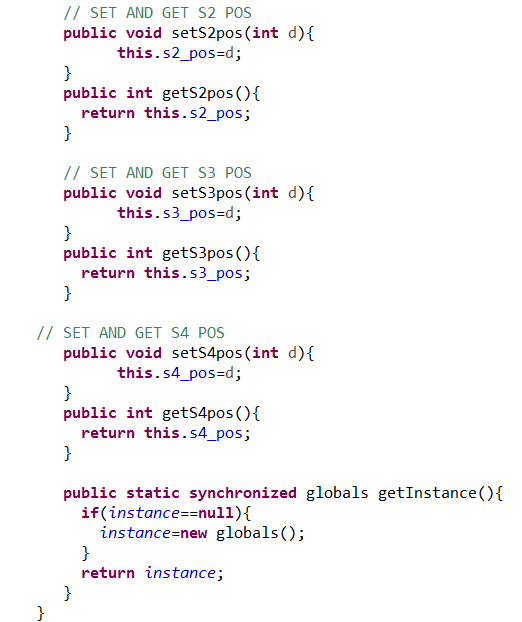








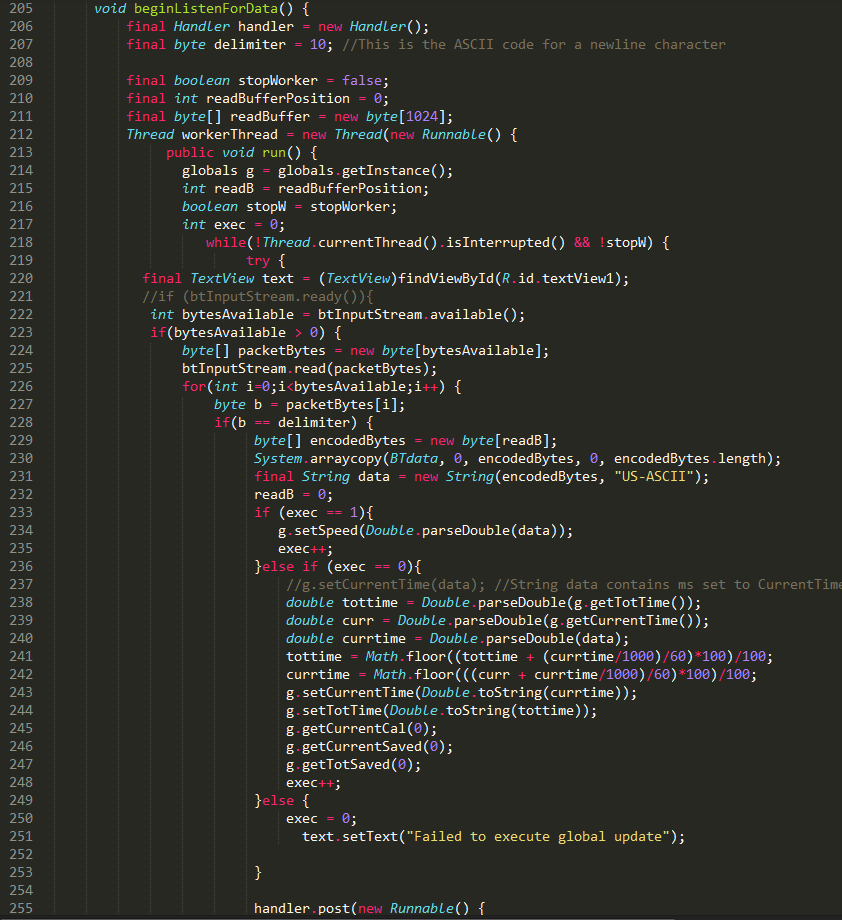


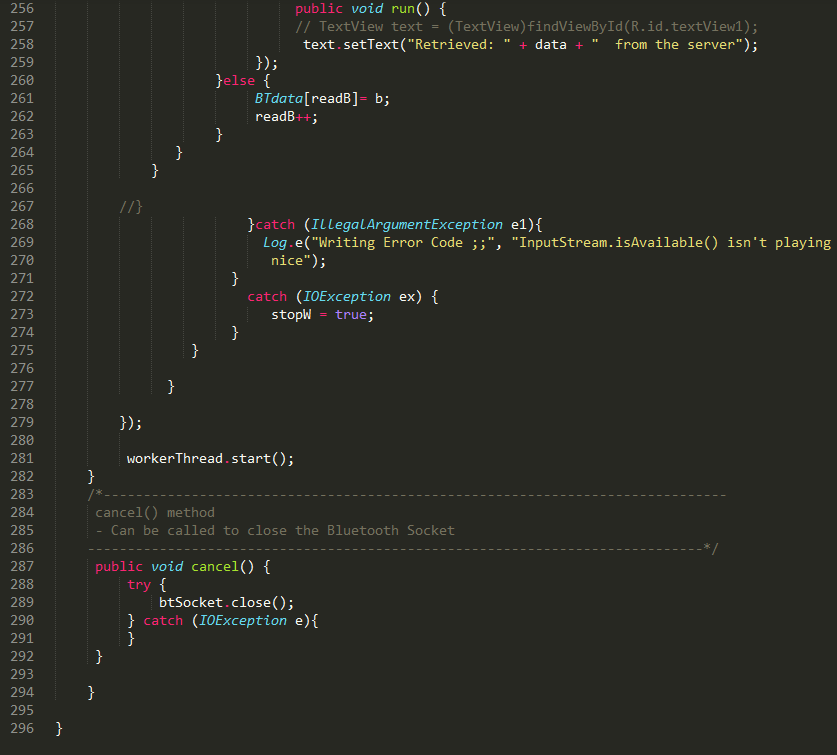


**BaseActivity.java** – location for all of the Bluetooth and data transfer implementation

This is where all of the data transfer comes from, implementing a call on the local Bluetooth adapter to connect to a specifically named Arduino “HC-06” and write a pre-determined value such that the Arduino can see the socket and return its values of time and revolutions with a separator character.







**Appendix 2 – Microcontroller**

**Code**

/\*-----( Import needed libraries )-----\*/

#include <PinChangeInterrupt.h>

/\*-----( Declare Constants and Pin Numbers )-----\*/

#define LED\_Left 13

#define LED\_Right 12

/\*-----( Declare Variables )-----\*/

////setup buttons

int PB\_Left\_int = 8;

int PB\_Right\_int= 7;

///////////////State of buttons

boolean buttonPushedLeft = false;

boolean buttonPushedRight = false;

////////Bluetooth///////////

char INBYTE;

unsigned long oldTime = millis();

unsigned long newTime= millis();

unsigned long elapseTime;

long sendrev;

long prevrev;

bool request;

bool conn;

int wait;

//////Hall Effect Sensor///////

long rev;

long old\_rev;

volatile int old\_time; // in order to store the time value of the current state, to be used in next state

int perimeter;// pi\*diameter

int diameter; // diameter of the tire

///////////////////////////////end of variables

void setup() {

//Turning Lights//

//set OUTPUTS

Serial.begin(9600);

pinMode(LED\_Left, OUTPUT);

pinMode(LED\_Right,OUTPUT);

pinMode(PB\_Left\_int, INPUT);

pinMode(PB\_Right\_int, INPUT);

//attach interrupt

attachPinChangeInterrupt(0,ISR\_left,RISING);

attachPinChangeInterrupt(23,ISR\_right,RISING);

//////////////////

//Bluetooth//

long sendrev=0;

long prevrev=0;

bool request=false;

bool conn=false;

int wait=0;

//////////////////

//Hall Sensor//

attachPinChangeInterrupt (18, magnet\_detected, RISING);

old\_time = 0;

perimeter = 0;

diameter = 20;

old\_rev = 0;

rev = 0;

//////////////////

}

void loop() {

///////////////////HALL SENSOR/////////////////////////

if (rev != old\_rev){

old\_rev = rev;

}

///////////////////////////////////////////////////////

///////////////////BLUETOOTH///////////////////////////

request = msgReq();

//conn=request;

msgSend(request);

///////////////////////////////////////////////////////

////////////////////TURNING LIGHTS/////////////////////

int buttonStateLeft = digitalRead(PB\_Left\_int);

int buttonStateRight= digitalRead(PB\_Right\_int);

if (buttonPushedLeft == true){

blink\_left();

}else {digitalWrite(LED\_Left,LOW);}

if (buttonPushedRight == true){

blink\_right();

}else {digitalWrite(LED\_Right, LOW);}

///////////////////////////////////////////////////////

}

/\*-----( Functions )-----\*/

//////////////////////////TURNING LIGHTS////////////////////////////////////////

//Interrupts

void ISR\_left(){

buttonPushedLeft = !buttonPushedLeft;

}

void ISR\_right(){

buttonPushedRight = !buttonPushedRight;

}

///////////////////////////////////////////////////

void blink\_left(){

// blinking loop

digitalWrite (LED\_Left, LOW);

delay (500);

digitalWrite (LED\_Left, HIGH);

delay (500);

}

void blink\_right(){

// blinking loop

digitalWrite (LED\_Right, LOW);

delay (500);

digitalWrite (LED\_Right, HIGH);

delay (500);

}

////////////////////////////////////////////////////////////////////////////////

//////////////////////////////BLUETOOTH/////////////////////////////////////////

bool msgReq() {//wait for command to be sent

INBYTE = Serial.read();

if (INBYTE == '1') {

return true;

}

else {

return false;

}

}

void msgSend(bool request) {

if (request) {

//Get the time elapsed

newTime= millis();

elapseTime= newTime-oldTime;

//number of revs since last request

sendrev=rev-prevrev;

prevrev=rev;

//reset time counter

oldTime=millis();

//send data

Serial.print("Time ");

Serial.println(elapseTime);

Serial.print("Revs ");

Serial.println(sendrev);

Serial.print("total ");

Serial.println(rev);

}

// if(!conn){

// wait++;

// Serial.println(wait);

// if (wait==10){

// wait=0;

// }

// }

request=false; //reset request to false

}

////////////////////////////////////////////////////////////////////////////////

/////////////////////////////HALL SENSOR////////////////////////////////////////

void magnet\_detected(){

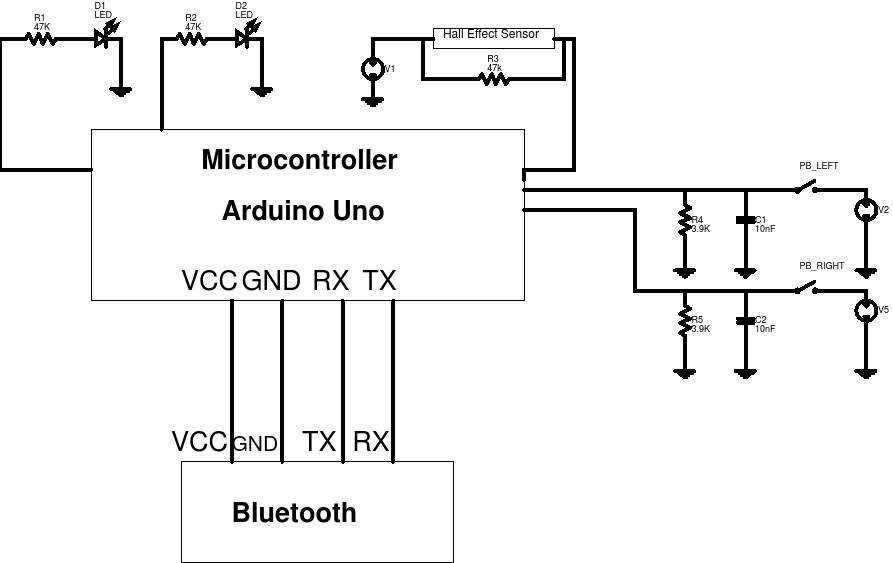
rev++;

Serial.println(rev);

}

////////////////////////////////////////////////////////////////////////////////

**Circuit Diagram**



**Block Diagram**

