# Final Report

Team TrakPak

Bernard Floeder, Oliver Hanes, Michael Lindt & David Surine

**Table of Contents**

# 1 Introduction

**The Problem**

At the end of last semester, we were tasked to form teams with our peers and brainstorm projects and products that will help make people’s lives easier. More specifically, we were asked to think of ideas to add utility to an already existing technology by means of wireless communication via the Internet of Things. This requirement brought with it many challenges. Not only would it be a technical challenge to incorporate Wifi into our project, but there was a challenge in producing a viable project idea that incorporated WiFi. While it would be awesome to incorporate WiFi into a toaster, there is only so much usefulness that an Internet-enabled toaster could give you. We wanted to make sure that, whatever the project idea we chose, we chose one that was both feasible for us to actually make, and worthwhile for us to actually make.

Many of us brought to the table ideas that incorporated passions for music, fitness, and practicality in many different combinations. However, most of our projects ended up being objects that only covered a small demographic and were more skin deep. It was indeed difficult to find objects that would provide new tangible benefits to a consumer if they were WiFi-enabled. Thus, we changed our approach for defining our project. Instead of focusing on the objects, we decided to focus on the problem itself. What are some problems that people have today that they cannot solve (or could be solved in a better way)? This switch in thinking turned out to lead to the breakthrough that led us to our final project. After some further brainstorming, we finally uncovered a problem that we felt would be appropriate to attack with a WiFi-enabled Senior Design project. The problem that we decided to solve with this project is the everyday lapse of memory that leaves virtually every college student in sporadic times of crisis, in which they are missing items essential for everyday tasks.

On any given day, it is perfectly possible for someone to go to class for an exam, take out their calculator, do poorly on the test, and storm out of the room. In their irritation, they forget their calculator on the ground next to where their backpack was. Likewise, it is equally possible for a person to take out their water bottle on a hot Friday afternoon for their 3:30 lecture and be in such a hurry to get to their room for a nap that they leave their water bottle lying beneath their seat in 101 DeBartolo. Of course, there is always the timeless practice of retracing steps across campus, both mentally and physically. However, this tedious process takes away precious minutes that can be used for studying or spending time with close friends outside of study circles. Moreover, items that are capable of holding enormous amounts of data (i.e. tablets, smartphones, external hard drives and laptops) carry with them incalculable consequences for school projects and businesses when they are misplaced. Finally, backtracking your steps is not always a very successful process! We wanted to eradicate the practice of backtracking as best we could and help people hold on to the things that make their world work.

Given our new problem, we began looking for any products currently on the market that are meant for helping people keep track of their everyday items. While there are some technologies out there that can help you keep track of your phone or keys (Find My iPhone, Tile, etc.), we were unable to find a technology whose sole purpose is to actively keep track of multiple items at a time in a convenient manner. This seemed like a hole in the market to us. Wouldn't it be great if there were a product that was designed to keep track of any and all of the items that you considered to be valuable? And wouldn't it be great if this product incorporated WiFi, such that there was a real convenience in the way that you kept track of all of your belongings? Since the answers to these questions were a definite "Yes!" for us, we decided that venturing further into this concept would be a worthwhile project that could actually help people solve a real problem as well as definitely incorporate the required IoT technologies as mandated by the Electrical Engineering Department.

Another important detail that we wanted to address was the market for which that we were designing our product. We chose to design for the market that we best knew: college students. Not only are college students often forgetful, but we know that modern college campuses are generally blanketed (or at least largely covered) by WiFi. Thus, given the constraints on our project, our own personal experience, and the problem that we wanted to solve, college students became the perfect target market. This recognition led to a few new specifications. We wanted to help each individual student without forcing them to pay an exorbitant amount of money for the convenience. College students, generally speaking, do not have tons of money to spend. Thus, we realized that we would have to do our best to find reasonable compromises between cost and functionality with our project.

Finally, although we realize that the problem of forgetting belongings is rather "first-world" and does not really help those in developing regions of the world, we believe that a well-conceived and well-executed project would have the potential to eventually reach the world at large. The problem itself is quite simple, but our design would need to be clever enough to bridge the gap between the major inconvenience of losing an object and the helpful convenience of having each individual item's last known location in your back pocket. Therefore, we felt that it was important to have the needs of everyone in mind when designing our solution, even though our project would definitely be catered to college students in America.

## The Solution

After much brainstorming, we finally settled on an idea that fit all of the above criteria and would solve our problem: a backpack. A backpack is an object that nearly every college student already needs and buys. A backpack is also an object that already carries most of one's other important objects (computer, computer charger, books, calculator, water bottle, etc.). Thus, if we could find a way to redesign the backpack, such that it used WiFi to alert you as to the status of your belongings in some way, then we would have found a very elegant way to solve our problem. This is the idea that became our Senior Design project, and we decided to call it "The TrakPak".

At the highest level, our prototype product is a backpack that has mounted hardware that has three major functionalities: the capability to "scan" items in and out of the backpack, a GPS chip to provide time and location stamps, and a Wifi chip to communicate that information to the user. In particular, we used a Wifi chip that could send the information to an MQTT server. From that server, we are able to gather that information with a mobile phone application, which can then format the information into an aesthetically appealing environment within an app. Our TrakPak app welcomes any user to name each new item scanned and see when and where they last had each item. That being said, each item that you wish to keep track of will need its own RFID tag with its own passive frequency to be scanned and its own serial number to be punched into the mobile app for tracking. Each tag costs next to nothing, so the bulk of the cost of the project will come from the hardware that will implement the software we develop for communication.

## Project Success and Functionality

We are very pleased with how our product has turned out and fully endorse its use as a functional prototype that could possibly even stand up against a team of patent lawyers. We believe that our work combines the aspects of our goal very fluidly and are proud with the aesthetic that we have come up with. The hardware is all fully functional and is able to scan in an RFID, give the user an auditory cue for successful scans, gather GPS data unique to that scan, and transmit all of this information via the WiFi chip to an MQTT server where it can then be picked up by the Mobile Application. The GPS is more finicky than we expected when it comes to providing GPS data indoors, however it is still able to ballpark the item close enough that we believe the location accomplishes its task of simply aiding someone in remembering where they left and item. And, while the app is only functional to the point where a person has to manually enter the serial data unique to each RFID tag, it is still able to keep a running history of each scan of each item and the of the backpack itself. It is able to visualize the location of each individual item via Google Maps API and even give directions on the shortest route that a person can take to each item. In all, we believe that we have created a prototype that even in its crudest stages can help the average college student cut down on lost time looking for items that should always travel with them. While our product cannot guarantee that a person will never lose any of their things during a normal day of class, we believe that the problem has largely been helped if not eradicated.

We are hopeful that this design has immense implications for product development on a macroscopic scale. That is, we believe this system of tagging items can be taken and developed much further by existing technology giants and can be applied to a number of social realms. If we were to replace the WiFi transmitter with a device capable of transmitting cellular data between the backpack and the mobile app, the backpack could be taken anywhere, not just within the confines of the WiFi fields of college campuses everywhere. For example, the backpack instead could be taken on hiking trails and more remote areas on the backs of more adventurous users. Moreover, if we were to be able to develop a cost effective way of tracking each item via bluetooth, then we could keep track of items in the backpack regardless of whether they were physically scanned right before placing them in the pack. The best part is that the concept doesn't even have to stay within the confines of a backpack but can extend to general personal belongings in satchels, purses, duffle bags, luggage and briefcases. It's almost surprising that we have not seen any products similar to ours with so many possibilities for making people's lives a little bit easier.

# 2 Detailed System Requirements

### 2.1 Embedded System Requirements

- PIC Microcontroller

    - Choose from PIC family because of familiarity with components/programming
    - All required passive components (Vcap, Decoupling Capacitors, etc.)
    - Programming Requirements
        - Language
            - C
        - Environment
            - MPLAB IDE
        - Hardware
            - PICKit 3
            - Pins on board for PICKit 3
- General Requirements
    - Must be able to communicate with all auxiliary hardware, including but not limited to:
        - GPS chip
        - Power Hardware
        - WiFi chip
        - RFID Scanner
    - Must have at least 4 available interfaces for proper serial communication with peripherals
        - Each auxiliary hardware piece must have an available communication protocol (UART, SPI, I2C, etc.)
    - Must have at least 3 I/O pins for controlling power

**2.2 Power Requirements**

- General Requirements
    - Input: Must be rechargeable via USB connection or connection to wall
    - Output: Must be able to connect to our microcontroller via a microUSB connection
    - Weight Requirement
        - As light as possible considering that the user will have to carry it around
        - Maximum: 5 lbs
    - Size Requirement
        - Shape of battery must not be bulky such that integration into the backpack is difficult and/or the backpack is not as usable
    - Total Maximum Power Draw: ~500 mA

        - The above power estimate has been made by choosing a likely candidate for each component and then determining its maximum power draw. This should give us a worst-case scenario power estimate.
            - Microcontroller
                - PIC Datasheet
                - Maximum Input Current = 300 mA
            - GPS chip
                - Gtop GPS
                - 30 uA (standby current) at 3.3V = 54 uW
            - LED Hardware
                - Average Diode Forward Current = 150 mA
                - Depends on current limiting resistor choice
            - WiFi chip
                - ESP8266
                - 15 mA typical sleep value, 170 mA TX max, 50 mA RX max
            - RFID Scanner
                - ID-12LA Datasheet
                - 35 mA maximum
    - Required Runtime: 12 hours
    - Required Battery Estimate: 7200 mAh

### 2.3 Wireless Interface Requirements

- General Requirements
    - Must be able to communicate with WiFi chip connected to the central PIC board and RFID scanning device
        - Must be able to connect to any unsecured, public networks
    - Phone system and WiFi chip both must be able to communicate within the domain of the created mobile app

### 2.4 User Interface Requirements

- General Requirements
    - Must be able to alert the user that each item has been successfully or unsuccessfully scanned
        - This will be done with a beep on the RFID board

    - Must have a functioning mobile app to provide scanned information for each item (Object, GPS & Time Stamp)

### 2.5 Installation/Use Requirements

- General Requirements
    - The TrakPak will not be user-installed; we will install and assemble all components.
    - The passive RFID tags will be installed by the user onto belongings of their choice via stickers.
    - The following components should be able to be installed directly on or connect directly to a PCB of dimensions ~4 in x ~3 in.
        - Microcontroller (and all its required passive components)
        - Clock Circuitry
        - GPS Chip
        - WiFi Chip
    - The RFID scanner must be installed in an easy-to-reach area of the interior of the backpack to protect it from weather.
    - The battery and microchip must be installed in a convenient location within the backpack to prevent damage to the battery and preserve interior space.
        - The battery's power cord must have a way to be secured when not in use.
    - All wiring must be secured and well hidden to prevent damage from wear and tear as well as preserve a sleek look.

### 2.6 Safety Requirements

- Must ensure that the battery can account for overcharging without overheating too much.
- Ensure that the battery charging process is well protected to avoid delivering shocks to the user.
    - Only possible dangerously high voltage is from charging the backpack off the wall.
- As a backpack, do not overburden yourself to avoid damage to spinal system

### 2.7 Mechanical Requirements

Our project is designed to improve a user's experience with a backpack. As such, there are several main mechanical requirements that cannot be exceeded or the TrakPak would cease to be an advantageous product. Primarily, the TrakPak must be able to be fully functional inside of a backpack without compromising the carrying capabilities of a comparatively sized backpack. In order to meet this expectation, we will have to use hardware that is small, lightweight, and durable.

The first consideration is size. Our hardware will need to be small enough to be incorporated into an existing backpack without taking up too much space. The TrakPak would be useless if there was no room to put anything into it. Therefore, we will need to buy parts that will minimize space while keeping effectiveness high. The major challenge for this consideration will be the power supply, since most power supplies are relatively large compared to the average microcontroller. The specific size requirement will be determined by the backpack that we choose to augment, however as a general rule we will need to buy the smallest parts possible that fit our requirements.

The second consideration is weight. Backpacks quickly become uncomfortable when they are filled with too much weight. While we may not be able to decrease how much a user tends to put inside the TrakPak, we can control how much the base weight is for an empty TrakPak. In order to keep this base TrakPak weight at a reasonable level, we will need to find parts that have minimal weight. Most of our components, like the microcontroller or the LEDs, will be of negligible weight. However, the power supply may be another challenge as batteries with the power capacity that we will need tend to be comparatively heavy. The specific base weight requirement for the TrakPak will be highly dependent on the weight of the backpack we choose to augment, however our other components should be under five pounds altogether.

The third consideration is durability. The TrakPak would be a highly ineffective product if it couldn't stand up to the rigors of daily life. Few backpack users treat their backpacks with special care. They are often dropped on the ground, stepped on, or in other ways misused. We cannot assume that the TrakPak will be treated any differently than a regular backpack by a user, and so our components will need to be able to survive these daily abuses. There are two ways we can meet the durability requirements. The first is by purchasing already durable parts. Some parts, like the microcontroller and its immediate plugins will be inherently fragile, but other parts, like the power supply or the RFID scanner, may have variable durability. The second way to meet our durability requirement is by packaging our parts. This packaging will need to meet the size and weight considerations, as well as sufficiently protect our hardware from the daily wear and tear of backpack usage.

In conclusion, the TrakPak is dependent on three mechanical requirements. Firstly, the hardware must be small enough to not hinder the TrakPak's carrying capacity. Secondly, the hardware must be light enough to not inhibit the user's comfort. Thirdly, the hardware must be protected so that the TrakPak's technological components remain functional. These three mechanical requirements will ensure the longevity of the TrakPak functionality and the user's satisfaction with the physical construction of the TrakPak.

# 3 Detailed project description

## 3.1 System theory of operation (how the whole thing works)

This section will detail how the entire TrakPak works in full detail. First, we want to define the vocabulary that we will be using. The TrakPak system is broken down into two main subsystems: the "TrakPak" subsystem, and the Mobile Phone App subsystem. By "Trakpak" subsystem, we are referring to all the components that are housed within the backpack, and the backpack itself. By "Mobile Phone App" subsystem, we are referring to just the Android-based Mobile Phone App. The TrakPak subsystem is further broken down into five subsystems: the RFID subsystem, the GPS subsystem, the WiFi subsystem, the Microcontroller subsystem, and the Power subsystem.

Now, we will explain the individual task of each subsystem, and their relation to one another. For reference, see Section 3.2 (System Block Diagram) to visually see the subsystems and connections t5hat we are talking about. The "brain" of the TrakPak system is our Microcontroller subsystem. The microcontroller controls when each of the other TrakPak subsystems (RFID, GPS, WiFi) receive power. The microcontroller also requests data and sends data to each of tho se subsystems as necessary. Each of those subsystems, in general, has one specific task.

- RFID - Receive object scans and send that information to the microcontroller
- GPS - Acquire GPS data and send that information to the microcontroller
- WiFi - Send the information that is gathered by the microcontroller to the Mobile Phone App, via an MQTT server

- Power - Provide regulated power to all other TrakPak subsystems (via a battery) and provide a way to charge that battery

The Mobile Phone App, in turn, has the general task of receiving the information sent by the WiFi module, via the MQTT server, and presenting it to the user in an intuitive and helpful manner. These are the general tasks of each subsystem. Now, we will explain how the each of these subsystems works together to give the TrakPak its functionality.

The main functionality of the TrakPak is its ability to “scan” a user’s object, and then tell the user what the object is, where it was scanned, and when it was scanned. The TrakPak accomplishes this task in the following way. Note that further technical details of how exactly each subsystem is working will be covered in the following sections of this document.

1. The entire TrakPak system is powered up, via the Power subsystem and the battery.
2. The microcontroller turns on power to the GPS subsystem and initializes it.
    a. The microcontroller controls power to the GPS subsystem by toggling a PMOS based switch on/off via one of the its digital output pins.
    b. The microcontroller confirms that the GPS is on by receiving an acknowledgement from the GPS.
    c. The microcontroller sends a command to the GPS to format its output data in an appropriate way. It receives an acknowledgement that this command has been received and processed successfully.
3. The microcontroller then turns on power to the RFID.
    a. This power is controlled by the same type of PMOS switch, regulated by a different digital output pin.
4. Now that the RFID is powered up, it waits indefinitely until the user scans an object.
    a. This “scanning” occurs via an RFID reader mounted in the backpack, and small RFID tags that are placed on items of interest (calculators, notebooks, water bottles, etc.).
    b. The user has to bring the the RFID tag very close to the RFID reader for the tag to be detected, in much the same way as one would scan a badge in an office building.
5. Once a scan is detected, the RFID reader will send the tag’s unique RFID tag number to the microcontroller.

    a. RFID tag numbers are 12 digit alphanumeric strings, unique to each specific tag.
    b. The RFID reader sends this string to the microcontroller via UART communication protocol.
    c. After the RFID tag has been successfuly read by the microcontroller, the RFID subsystem is powered off to preserve power and prevent the user from scanning any new tags until the current one is processed.
6. Once the microcontroller has received the RFID tag, it checks the last known GPS data point.
    a. The GPS subsystem operates via a timer interrupt function. Every two minutes, this interrupt function triggers and polls the GPS subsystem for its most recent data collection.
        i. If that GPS information is valid (we have a new location fix), the microcontroller updates its GPS data point.
        ii. If that GPS information is not valid (we are inside and cannot get a fix), the microcontroller will use its most recent valid data point.
    b. This communication between the microcontroller and GPS subsystems occurs via UART communication protocol.
    c. The GPS data that we used is in GGA format. For a detailed description of what this format looks like, see the GPS subsystem section and the appendix.
        i. Note that the GPS data also provides UTC (Coordinated Universal Time) data. Thus, the GPS data includes both when and where something has been scanned.
7. Once the microcontroller has the RFID and GPS data, it organizes that data into a convenient string for sending to the WiFi.
    a. This process is done is software. The code within the appendix shows exactly how this part works.
    b. The reason that this step is important is because we needed to organize the data in such a way that the Mobile Phone App can easily parse it.
8. The microcontroller powers on the WiFi subsystem and then sends this organized string to the WiFi subsystem.
    a. The microcontroller controls power to the WiFi subsystem via the same type of PMOS switch, which is controlled by a different digital output pin.

    b. This communication between the microcontroller and the WiFi module occurs via UART communication protocol.
    c. Before actually sending the data to the WiFi, the microcontroller waits for confirmation from the WiFi that it is on and connected to WiFi.
    d. The organized string that contains the unique RFID tag number and GPS time and location information is then sent.
    e. The microcontroller then waits for another confirmation that the WiFi module was able to successfully send the data to the MQTT server.
    f. Once the microcontroller has confirmation the information was successfully sent, it powers off the WiFi subsystem.
9. Once the WiFi module has uploaded the data to the MQTT server, the TrakPak system is essentially finished, and the Mobile Phone App comes into play.
10. The app subscribes to the MQTT server and waits for the TrakPak to upload new scan data.
    a. The user has the ability to connect/disconnect from the MQTT server via a button on the screen.
11. Once the app sees that new data has been uploaded, it processes that data and presents it in a few ways. For more detailed information on exactly what the app is doing and what it looks like, see Section 3.4 (Mobile App).
    a. The app parses the data to separate the RFID tag, the time data, and the location.
        i. The app gives the user the option to rename those alphanumeric strings to something more useful such as “water bottle” or “notebook”. Otherwise, it just displays the alphanumeric string.
    b. The app displays a running log of what was scanned, where it was scanned, and when it was scanned.
    c. The app also has a map feature, such that if a user taps a particular scanned event, a standard mobile application map comes up with a pin dropped at the location of that scan.
12. The action is now finished. The TrakPak will have reset to Step 4, where the RFID system is powered up and awaiting the next scan. The Mobile App will display past information and await the next update to the MQTT server.

Following the steps above, we have showed how the TrakPak accomplishes its core functionality. In addition, there are a few other actions that the TrakPak also performs that add to its usefulness and robustness.

In addition to updating the location and time of a scanned object, the TrakPak has the ability to periodically update the location of the TrakPak itself. Below is a description of how this action is performed.

1. As mentioned above, the microcontroller incorporates a timed interrupt function that will periodically check and update the current GPS location. This same interrupt function is used to send the backpack's GPS information to the MQTT server.
2. Every 2 minutes, when the timed interrupt function triggers, the interrupt function will check and increment a variable. When the variable has reached a certain value (which occurs about every 6 minutes), the microcontroller will acquire the most recent valid GPS data and use that as the backpack's most current location.
3. Following Steps 8 through 12 from the main functionality above, the microcontroller sends the backpack's data to the WiFi subsystem, which sends the data to MQTT server. That information is collected by the app and displayed to the user.

This feature of sending the backpack's current location operates on the same principle as the main functionality, except that it is triggered by a timed interrupt instead of a scanned object event.

Finally, the TrakPak has the ability to save scanned data, in the event that we are not in WiFi range when the scanning occurs. The TrakPak will then check periodically if WiFi is available, and if so, send those saved scans to the user. Below is description on how this process works.

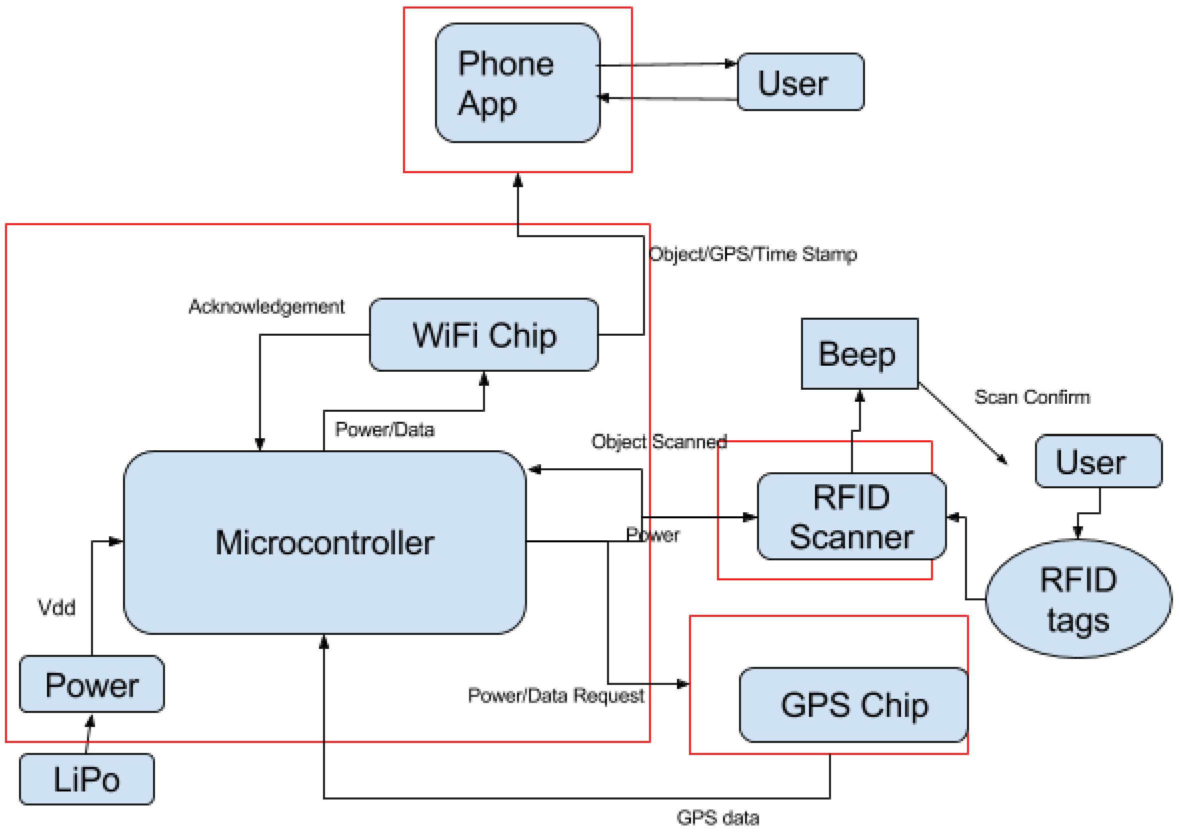
1. Looking up to the main functionality description, there is a point (8.c.) in which the microcontroller waits for the WiFi subsystem to confirm that it is connected to WiFi. If the WiFi sends back that it is not connected, or if the WiFi subsystem times out and does not respond for about 30 seconds, then this feature will be implemented.
2. The software will then save the data that would have been sent (RFID tag, time and location) in an array in software.

    a. The current implementation has two such arrays, meaning that we can only save of to two scans that occur outside of WiFi at a time. In theory, any arbitrary numbers are possible.

3. When the timed GPS interrupt function occurs, the software will check to see if any of the saved arrays have been populated by data. If one or both of the arrays have scanned data, the microcontroller will implement the same sending feature from the main functionality, starting from step 8.
    a. It will check for a WiFi connection, send the data via WiFi to the MQTT server, and the app will process and display the scanned data to the user.
    b. If there still is not a WiFi connection, the data will remain saved in the array, and the software will try again the next time that the interrupt function is triggered.
4. Below are a couple notes on this process.
    a. If the backpack tries to send its own location and cannot find a WiFi connection, that data will also be saved into an array until WiFi is found.
    b. If the TrakPak cannot send something via WiFi, and both save arrays are already filled with data, the TrakPak will overwrite the arrays with this new scanned data. Thus, in its current form, some scanned information can be lost!

In conclusion of this section, we have given the detailed flow of how the TrakPak accomplishes its main functionality of sending object information to the user. In addition, we have detailed the TrakPak's auxiliary functionalities of updating the backpack's location and saving scans when WiFi is unavailable. The detailed design of each subsystem are given below in the subsequent subsections of Section 3 of this document.

## 3.2 System Block Diagram



## 3.3 Detailed design/operation of subsystem 1: TrakPak

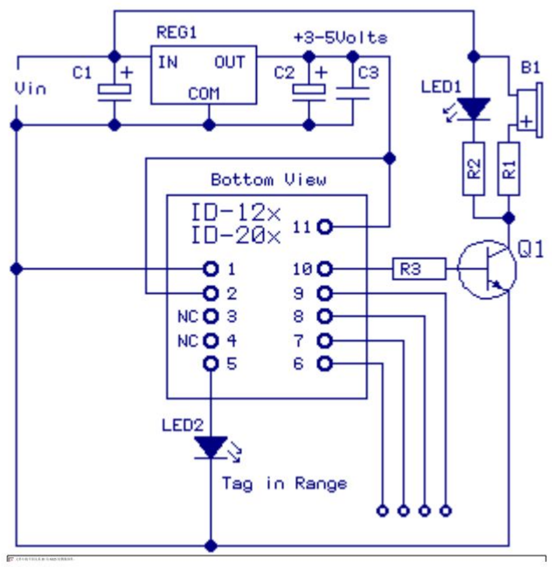
### 3.3.1 Detailed design/operation of TrakPak subsystem 1: RFID

Datasheet:
http://cdn.sparkfun.com/datasheets/Sensors/ID/ID-2LA,%20ID-12LA,%20ID-20LA2013-4-10.pdf

- Purpose:
  - To allow for the scanning of items
- Requirements:
  - Must be able to receive a scan from any passive RFID tag and proceed to output that RFID tag to the PIC microcontroller through serial communication.
  - Low current (<50mA)
  - Small package size (<4in^2)
  - Beep/LED to notify user of scan
  - Ability to read passive RFID Tags
- Inputs:
  - Power On/Off from Microcontroller
  - Data from RFID tags
- Outputs:
  - Data to Microcontroller
  - Data request to RFID tags
- Why this part was chosen:
  - The ID-12LA was chosen because for the TrakPak it is essential that users are able to scan items in and out of the backpack conveniently. This makes it important to have an RFID scanner that is compact, low current (under 50mA), ability to communicate serially, and have a decent scan range (approx 5cm). The ID-12LA was the commercial part that fulfilled all these requirements and was easily interfaceable to other subsystems.
- Communication protocol/programming:
  - Serial communication was used. No programming was required for this component, except to set up a UART so that our microcontroller and the transmit pin on the RFID device were talking on the same line with a baud rate of 9600.

**Schematic of RFID Module:**

### 8.1 Circuit Diagram for ID-12LA, ID-20LA



| Parts List | |
|---|---|
| **Part #** | **Value** |
| R1 | 100R |
| R2 | 4K7 |
| R3 | 2K2 |
| C1 | 10uF 25v electrolytic |
| C2 | 1000uF 10v electrolytic |
| C3 | 100nF |
| Q1 | BC457 or similar |
| LED1 | Read LED |
| LED2 | Tag In Range LED |
| B1 | 2.7khz – 3kHz 5v PKPK AC |

- The circuitry above is fairly simple and straightforward. Essentially there is a voltage that is input to pin 2, and this schematic shows a voltage regulator to accomplish this. We bypassed this by just outputting our already regulated voltage from our main board to pin 2 (3.3V DC). There is also LED circuitry on this schematic which will be turned on when pin 10 goes high, which will signify a scan has occurred. Also included on our board (and this schematic) is a beeper that will notify when a user has successfully scanned, also triggered by the mosfet on pin 10. The only other pins that we need to worry about on this schematic are pin 1 which is just connected to ground and pin 9 which transmits the RFID data to our PIC32 through a UART connection.

- How was this tested?
  - The RFID module was tested by writing a program to print out the serial information it reads to Terminal. Once an RFID card is scanned we should see it's unique RFID tag show up on terminal if it is working. Once we confirmed that this subsystem worked we were able to utilize it's functionality in the overall project flow.

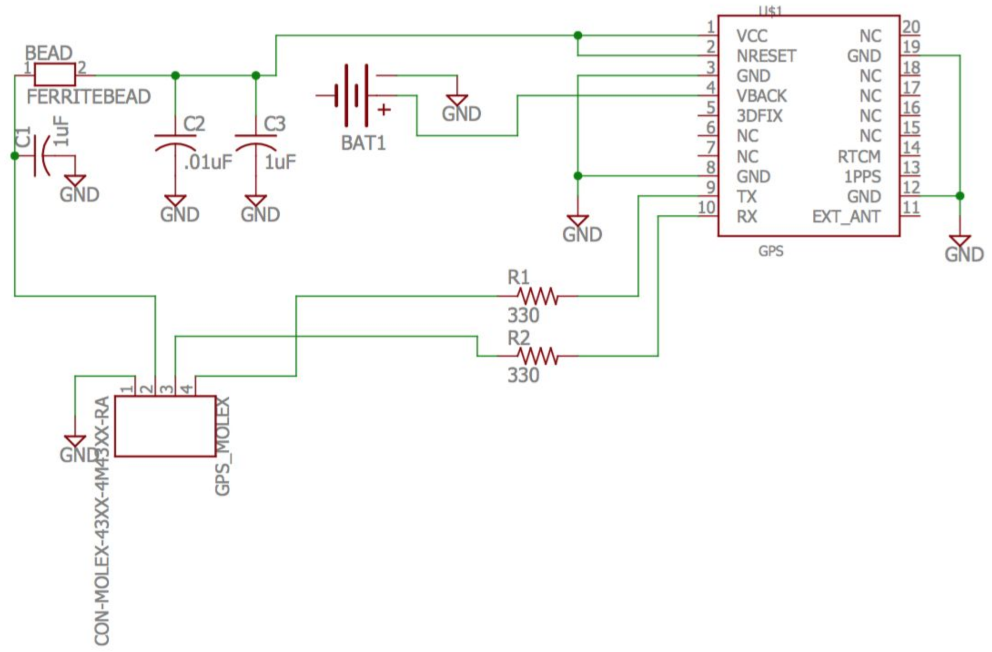
## 3.3.2 Detailed design/operation of TrakPak subsystem 2: GPS

Datasheet:
https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf

- Purpose:

    - To acquire GPS and real-time clock information upon request.
- Requirements:
    - Small package size to not take up too much space in the TrakPak
    - Low current (<25mA)
    - Give accurate latitude and longitude data +/- 30 feet
    - Give accurate real-time clock data
    - Send data via serial UART
    - Recognize MTK NMEA packet commands
- Inputs:
    - Data request from microcontroller via serial UART
    - MTK NMEA packet commands
- Outputs:
    - GGA Data
        - $GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65
        - UTC Time (064951.000, hhmmss.sss)
        - Latitude (2307.1256, ddmm.mmmm)
        - Longitude (12016.4438, dddmm.mmmm)
        - Other data not important for our project (1,8,0.95,39.9,M,17.8,M,,*65)
- Why this part was chosen:
    - The GlobalTop - FGPMM0PA6H GPS was chosen for this project simply because it is a GPS that met all of our requirements, it was simple to communicate to, and was a fairly commercial part with good documentation on it.
- Communication protocol/programming:
    - To actually program the GPS to be in different modes or states we needed to send MTK NMEA commands to the RX pin of the GPS. An example of this would be the command to tell the GPS to only give us GGA Data: $PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*2C. The GPS then would only send GGA data over the TX line to the microcontroller. We were using a serial UART to establish communication with a baud rate of 38400.

**Schematic of GPS Module:**



- The GPS module schematic consists of powering the gps, and then connecting the communication pins. As you can see there are 4 pins coming from the main microcontroller board via molex connections. To power the board we send 3.3V to the VCC pin with a few decoupling capacitors and a ferrite bead to reduce power noise and provide stability to the power system. We also have a coin battery on the board to provide backup voltage to the GPS so that it can retain its fix while in low power mode, and have the ability to hot start. Finally the TX and RX pins connect to serial pins in the microcontroller after going through a few damping resistors to reduce EMI.

- How this subsystem was tested:
    - In order to test that the GPS was working we made sure to establish a solid communication line between the UART pins of the microcontroller and the GPS. If this was done correctly we should be able to see the microcontroller send data to the GPS and the GPS send data back to the microcontroller. We hooked up a logic analyzer and tested it, and once we were able to confirm that this communication was working and that the GPS was sending the correct data, we were able to proceed with assimilating the GPS into our total program system

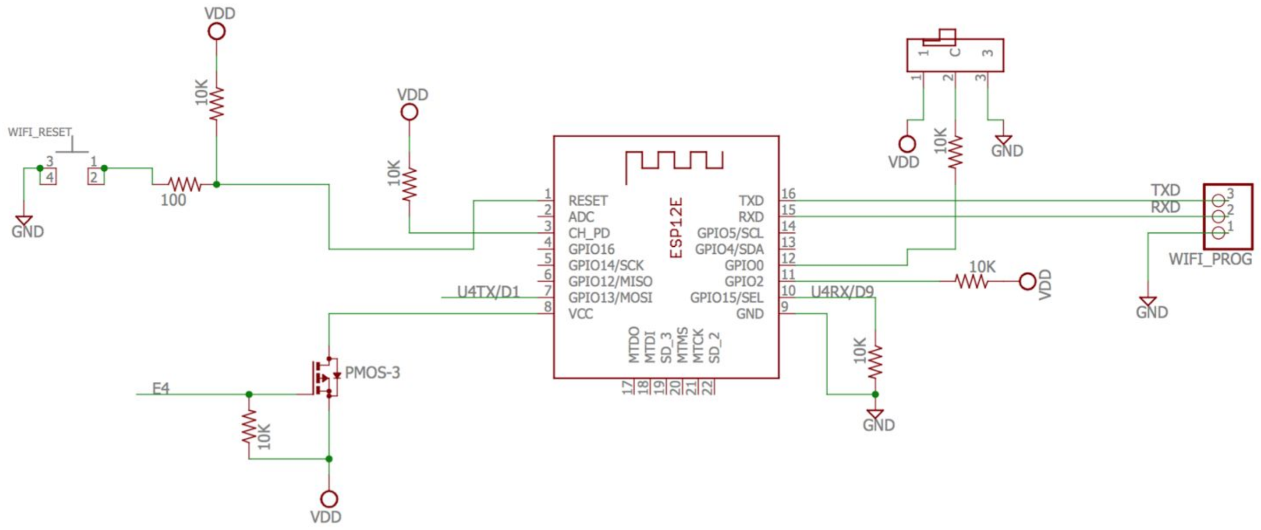
### 3.3.3 Detailed design/operation of TrakPak subsystem 2: WiFi

Datasheet:
https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf

- Purpose
    - To communicate between the User App and the Trakpak via Wifi
- Requirements:
    - Connect to unsecure wifi networks
    - Subscribe to MQTT Servers
    - Publish to MQTT Servers
    - Powered by 3.3V DC
    - Communicate via serial UART to transmit and receive data
- Inputs:
    - Data from Microcontroller (string to be sent to app)
- Outputs:
    - Data request from Microcontroller
        - Acknowledge that it is connected to wifi
        - Acknowledge that it has sent the data to MQTT Server
    - RFID Tag/GPS/Time Stamp to Phone App via wifi
- Why this part was chosen:
    - The ESP8266-12E was chosen primarily because it is the leading part in do it yourself internet of things projects (a requirement for this project), and it fulfills all the requirements that we needed.
- Communication protocol/programming:
    - We communicated to the ESP8266-12E from the microcontroller by using serial UART communication opening up a transmit and receive line with a baud rate of 9600. Through this serial line, we were able to send the ESP strings we wanted sent to the MQTT server and get acknowledgments back from the ESP that the sending completed.
    - Once the ESP received strings to send to wifi, it communicated over standard unsecured wifi networks to establish a connection with an MQTT server and then published to a topic we specified it too.
    - To program the ESP8266-12E we used the Arduino environment due to the heavy documentation on the subject already, and the ease of commands for our particular needs. All we needed to do in the program was establish wifi, connect to the MQTT server, read in a string through serial data, and then send it to a topic on the MQTT server. Utilizing the

ESP8266 and softwareSerial libraries on arduino this program was simple to write and easy to implement.

**Schematic of WiFi Module:**



- This circuit models the WiFi system and the various connections on the ESP-12E. To begin I will describe the more straightforward aspects of it. There is 3.3V going to VCC on the ESP (triggered by a mosfet opening up) to power the WiFi chip. Next the TXD and RXD pins are brought to a 3 pin header used to actually program the ESP. This allows it to be programmed serially. GPIO13/MOSI on the ESP is connected to the UART 4 TX pin on the microcontroller, and is used to receive information the microcontroller sends. GPIO15/SEL is connected to the UART 4 RX pin on the microcontroller which is used to send acknowledgements. GPIO15/SEL is also connected to GND because it is the protocol for programming the ESP. For this same programming protocol, GPIO2 is held high. The last aspect of programming the ESP is the GPIO0 pin, which specifies if the ESP is in programming mode or run mode during startup. If the switch is holding GPIO0 high the ESP is in run mode, and if low it is in programming mode. Finally we have a switch push button to reset the ESP device. The reset pin is held high until the switch is pressed to bring the pin low, thus resetting the ESP8266-12E.

- How this subsystem was tested:
    - The WiFi subsystem was tested in a few different steps. First we needed to test that we could connect to a wifi network and subscribe to a MQTT client. We tested this by using a blink program to blink the light if we send

    "blink" over the MQTT server. We were then able to verify that we were connected properly and able to communicate over the MQTT server.
  -
  - Following confirmation that our ESP device connected properly, we needed to make sure that it could send/receive serial data through a UART line to communicate with the microcontroller. We tested this by sending data to the ESP from the microcontroller, and then having the ESP send data back to the microcontroller once it was received. We confirmed this was working by studying the data lines with the logic analyzer, and confirming messages were being sent both ways.

  - Finally we needed to make sure the ESP could receive data and then send it to the MQTT server. We tested this by sending data to the ESP through the microcontroller and then verifying that the data was published to the MQTT Server which we could monitor on our computers.

### 3.3.4 Detailed design/operation of TrakPak subsystem 4: Microcontroller

Datasheet:[http://ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/61156H.pdf)

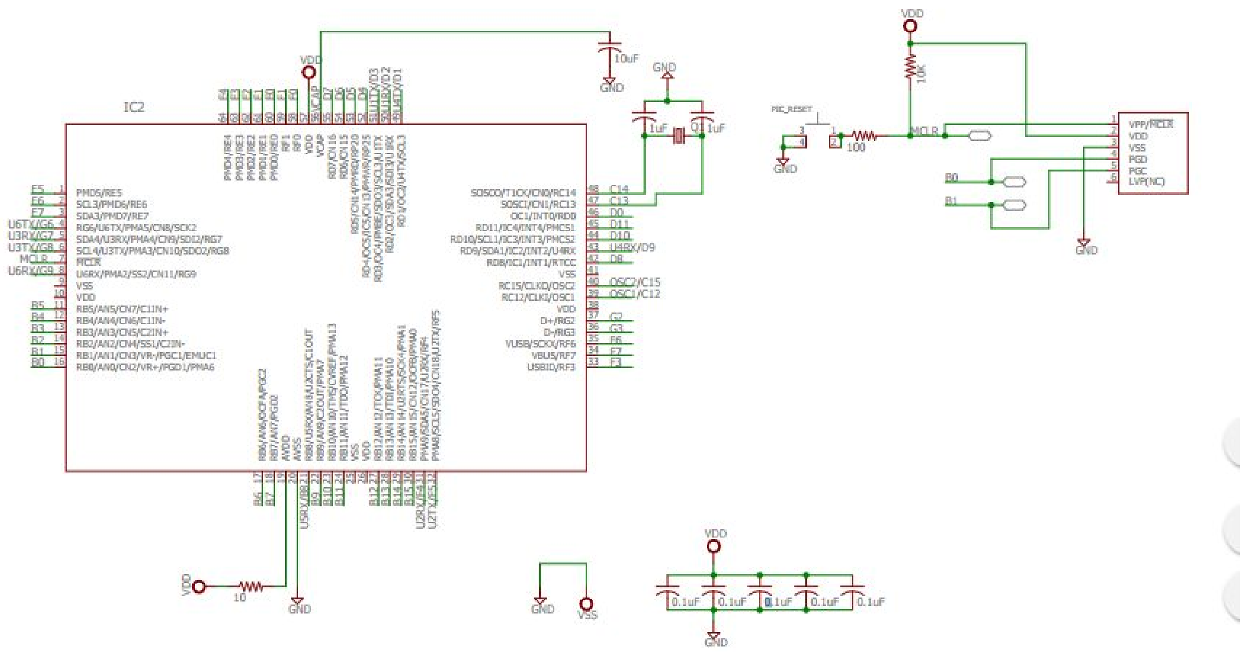
- Purpose
  - To coordinate all power and communication between the other TrakPak subsystems
- Requirements
  - PIC Microcontroller
    - Choose from PIC family because of familiarity with components/programming
    - All required passive components (Vcap, Decoupling Capacitors, etc.)
    - Programming Requirements
      - Language
        - C
      - Environment
        - MPLAB IDE
      - Hardware
        - PICKit 3
        - Pins on board for PICKit 3
  - General Requirements

        - Must be able to communicate with all auxiliary hardware, including but not limited to:
            - GPS chip
            - Power Hardware
            - WiFi chip
            - RFID Scanner
        - Must have at least 4 available interfaces for proper serial communication with peripherals
            - Each auxiliary hardware piece must have an available communication protocol (UART, SPI, I2C, etc.)
        - Must have at least 3 I/O pins for controlling power
- Inputs:
    - Vdd from Power Circuitry
    - RFID Scan Data
        - UART2
    - GPS Data
        - UART3
    - WiFi Acknowledgement Data
        - UART4
- Outputs:
    - Debugging Communication
        - UART1
    - GPS Commands
        - UART3
    - WiFi Send Data
        - UART4
    - MOSFET switch control signals
        - 3 digital I/O pins (E2, E3, E4)
- Why this part was chosen:
    - We first wanted to use a microcontroller from the PIC family because we were familiar with PICs from the first Senior Design class that we took. We knew that there were PICs that could satisfy our other requirements, and it would have difficult to learn how to work with a whole different microcontroller. We chose the PIC32MX695F512L for two reason.
        - It satisfied all of our other requirements. It had enough UART modules for our needs, it had other available digital outpins, and it would not consume too much power.
        - We were already familiar with this exact microcontroller from the first Senior Design class. We figured that, since there were not

drastically simpler options for us, we might as well use the exact microcontroller that we had learned how to use.

- Communication protocol/programming:
  - In order to program the microcontroller, we used the MPLAB IDE environment and wrote our programs in C. We used a PICKit 3 to program the device. Thus, our boards had the standard 6 pin connection that is required to use the PICKit3. This is a standard way to program the PIC32, and we had learned this method in the first half of the Senior Design class.

**Schematic of Microcontroller Module:**



- The schematic shows the major components of our microcontroller module. In the top left, we have the microcontroller itself. All power pins are properly tied to Vdd or GND appropriately, and Vcap is tied to a 1206 10 uF capacitor. We designed the microcontroller to operate off its secondary oscillator pins using a crystal in order to save power. In our actual implementation, we did not end up actually installing the crystal because power consumption was not an issue for our prototype. Instead, we used the internal RC oscillator as our clock source. We can also see that certain microcontroller pins are labelled. In the top left, we have E2, E3, and E4. These pins are the digital output pins that control power to the RFID, GPS, and WiFi modules respectively by controlling FET switches. Those switches are directly discussed in Section 3.3.6. Spread around the microcontroller, we can see that UARTs 1 through 4 are labelled because these are the main communication pins for the project. UART1 goes to debugging pins, UART2 to the RFID, UART3 to the GPS, and UART 4 to the WiFi module.

- In the top right, we have the programming pins. These pins include a Vdd and GND, a Reset pin tied appropriately to a push button, and two data pins for sending the program to the microcontroller. At the bottom, we have our five decoupling capacitors and ground. These components will be directly discussed in Section 3.3.6.

- How this subsystem was tested:
    - This subsystem was initially tested using the Dev board from Senior Design. Since the microcontroller was exactly the same, we could work on our programming for our project and be confident that the program would transfer over to our own custom designed PCB fairly well.
    - When we got to the point where we put our own PCB together, the first thing we did was to check and double-check all of the PICs pins, especially because the pins on the package we used (64 PIN QFN) are very small. We checked that power and ground were properly connected in all the right places, and then checked that our communication pins were all properly hooked up.
    - Once we were sure that all the pins were properly connected, the microcontroller was tested by incrementally making additions/changes to its program and seeing how it responded. This process was relatively quick because once we were saw the behavior that we expected, we were sure that the module was at least installed correctly. From there, we just had to change our code until the project worked as we wanted it too.

### 3.3.5 Detailed design/operation of TrakPak subsystem 5: Power

- Purpose:
    - To deliver proper power to all elements of the board
    - To allow a way to charge power
- Requirements:
    - General Requirements
        - Input: Must be rechargeable via USB connection or connection to wall
        - Output: Must be able to connect to our microcontroller via a microUSB connection
        - Weight Requirement

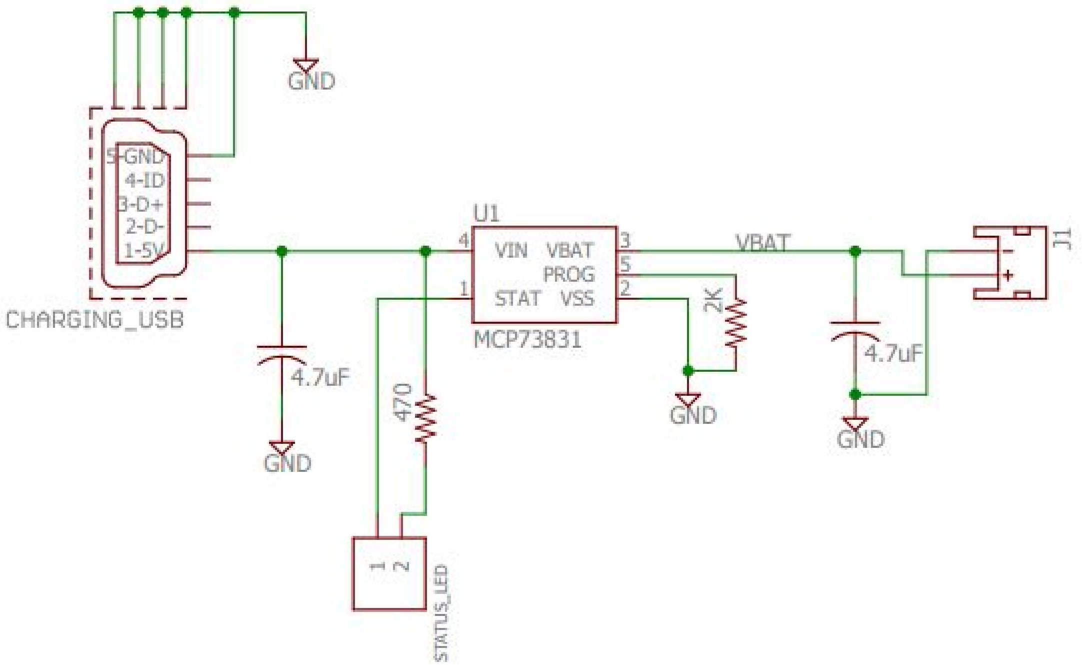
            - As light as possible considering that the user will have to carry it around
            - Maximum: 5 lbs
        - Size Requirement
            - Shape of battery must not be bulky such that integration into the backpack is difficult and/or the backpack is not as usable
        - Total Maximum Power Draw: ~500 mA
            - The above power estimate has been made by choosing a likely candidate for each component and then determining its maximum power draw. This should give us a worst-case scenario power estimate.
                - Microcontroller
                    - PIC Datasheet
                    - Maximum Input Current = 300 mA
                - GPS chip
                    - Gtop GPS
                    - Tracking Current = 20 mA
                    - 30 uA (standby current) at 3.3V = 54 uW
                - LED Hardware
                    - Average Diode Forward Current = 150 mA
                    - Depends on current limiting resistor choice
                - WiFi chip
                    - ESP8266
                    - 15 mA typical sleep value, 170 mA TX max, 50 mA RX max
                - RFID Scanner
                    - ID-12LA Datasheet
                    - 35 mA maximum
        - Required Runtime: 12 hours
        - Required Battery Estimate: 7200 mA
- Inputs:
    - Battery source
        - We chose a 5000 mAh LiPo battery with a JST connector
    - USB port for recharging the battery
- Outputs:
    - 3.3 V power to all of our components
    - Ability to source up to ~600 mA instantaneously at any given time

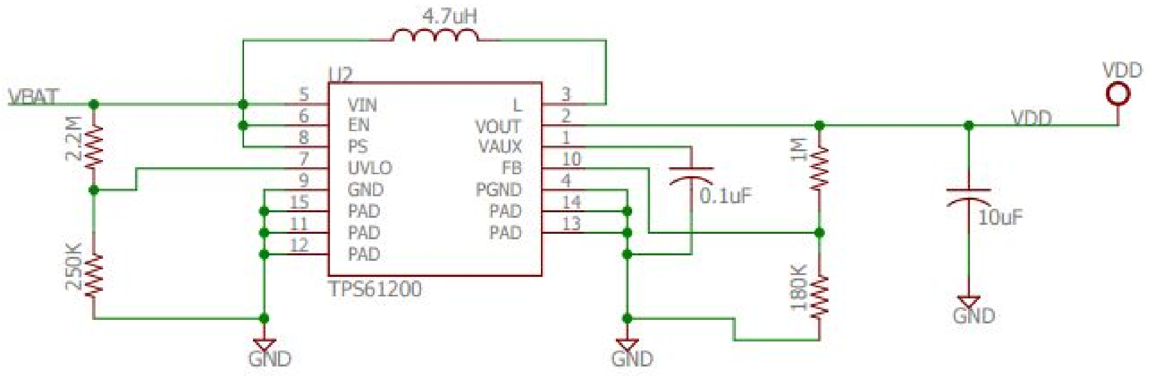
- Why this part was chosen:

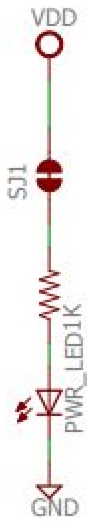
    - As far as components go, we had to choose three major parts: a battery, a charging IC, and a DC-DC converter of some kind. Below are those choices and the reasons for them.
        - Battery - 5000 mAh LiPo
            - Our battery estimates were based on a worst-case scenario of everything being on all the time, drawing maximum power, for 12 straight hours. Since we knew that we would program our components to be smarter than that and turn on only as needed, we knew we could undershoot our worst-case estimate. On the other side, we knew that this battery would be going in a backpack, so while we wanted to keep the weight of the battery down, we knew that it also did not have to be the smallest battery out there since the weight would be negligible compared to that of the backpack and its belongings.
        - Recharging - MCP73831
            - This IC has the ability to regulate the charging of a LiPo. We were exposed to it because we ordered the SparkFun LiPo Charger Booster as part of our initial testing to be able to charge the battery. Since that chip worked well, we borrowed from their design for our own board, and that included using the same charging chip.
        - DC-DC boost converter - TPS61201
            - This IC takes input power over anywhere between 2.4 to 5 V and delivered output power of 3.3 V. A similar chip (TPS61200) was used on the SparkFun LiPo Charger Booster to regulate the LiPo's output power, so we chose a closely related chip (TPS61201) that delivers a fixed output voltage of 3.3 V. Additionally, this chip has upwards of 90% efficiency for output currents of up to 300 mA. Since we knew that we would be turn modules on and off intelligently, we felt that this current and efficiency mark would be sufficient for our project.

- Communication protocol/programming:
    - None of these components require programming. We did have to use the datasheets to determine the appropriate resistors, capacitors, inductors, and board layout schemes for each of these parts. Those choices can be seen on our schematic and board files.

**Schematic of Power Module:**







- The top schematic shows the MCP73831 part of the design. We can see that Vin is connected to the Vdd line of a microUSB plug, which is properly grounded. This connection allows us to charge the battery via that microUSB plug. The STAT pin is connected to a 2 pin conenctor that we can opt to connect to an LED and resistor in order to alert the user whether the battery is charging or not. If the LED lights up, the battery is charging; if the LED is off, the battery is fully charged. VBAT is tied to the positive terminal of the JST connector that connects the battery to the board. The PROG pin is connected to a 2k ohm resistor to ensure that the chip charges the battery at its maximum rate of 500 mA. The VSS pin is tied to ground.

- The middle schematic shows the TPS61201 portion of the power circuitry. VIN is tied to VBAT, which comes from the positive end of the battery connection. VIN is also connected to the L pin via a 4.7 uH inductor, which was specified by the datasheet for proper operation. The EN and PS pins are pulled high to ensure that the chip is always functioning in normal mode. The UVLO pin is connected to a voltage divider between a 2.2 M ohm resistor and a 250k ohm resistor. This divider sets the cutoff input voltage below which the chip will shut itself down to prevent internal damage. Our resistor choices set the cutoff voltage value at about 2.5 V, according to the datasheet. The GND pins, PGND pin, and PAD pins are all tied to ground.  The FB pin is tied to another voltage divider. However, this was an error in design because we designed for the TP61200 chip, which requires a voltage divider on the FB pin because its output voltage can be adjusted. We ended up using the TPS61201 chip, which has a fixed output voltage of 3.3 V and requires that FB be tied directly to VOUT. Therefore, on our board, we replaced the 1M ohm resistor with a 0 ohm resistor, and left the 180k ohm resistor completely uninstalled. This change ensured proper operation on our actual board with the TPS61201 chip. VAUX is connected to a 0.1 uF capacitor, which goes to ground. Finally, Vout has a decoupling 10 uF cap on it, and becomes the main Vdd for the entire board.

- The bottom schematic simply shows the power LED that we designed on the board so that we would know that the board was receiving power. The solder jumper was added so that we could cut off power to this LED if we ever wanted to. Otherwise, it is a basic LED circuit with a current limiting resistor.

- How this subsystem was tested:
    - Before making our board at all, we had confidence in the parts we had chosen because we used the SparkFun board to charge and discharge the LiPo battery successfully. We knew that we would be making some changes, such as using the TPS61201 instead of the TPS61200, but we were confident in the general choices of our design.
    - When we got our own board, the first thing that we did was checked to ensure that Vdd and GND were not shorted out anywhere on the board. After confirming that, we installed our components and then again checked to ensure that we hadn't shorted Vdd and GND. Once we were fairly certain that all of our connections were properly made, we connected the LiPo battery. We saw that the power LED lit up as it should. We used a multimeter to confirm that the output of the DC-DC converter was 3.3 V. We then checked to make sure that all parts of the board that should be receiving power were, in fact, receiving it. We also checked various parts of the board that we knew should not be receiving power to make sure that they were not powered. Finally, we plugged in a USB for charging and checked the status LED pins to ensure that the LiPo was being charged.

### 3.3.6 Detailed design/operation of TrakPak subsystem 6: Misc.

- Purpose:
    - The purpose of this section is to account for all the smaller elements of our design that were necessary for proper operation, but did not fit nicely into any of the other subsystems.
    - Decoupling Capacitors

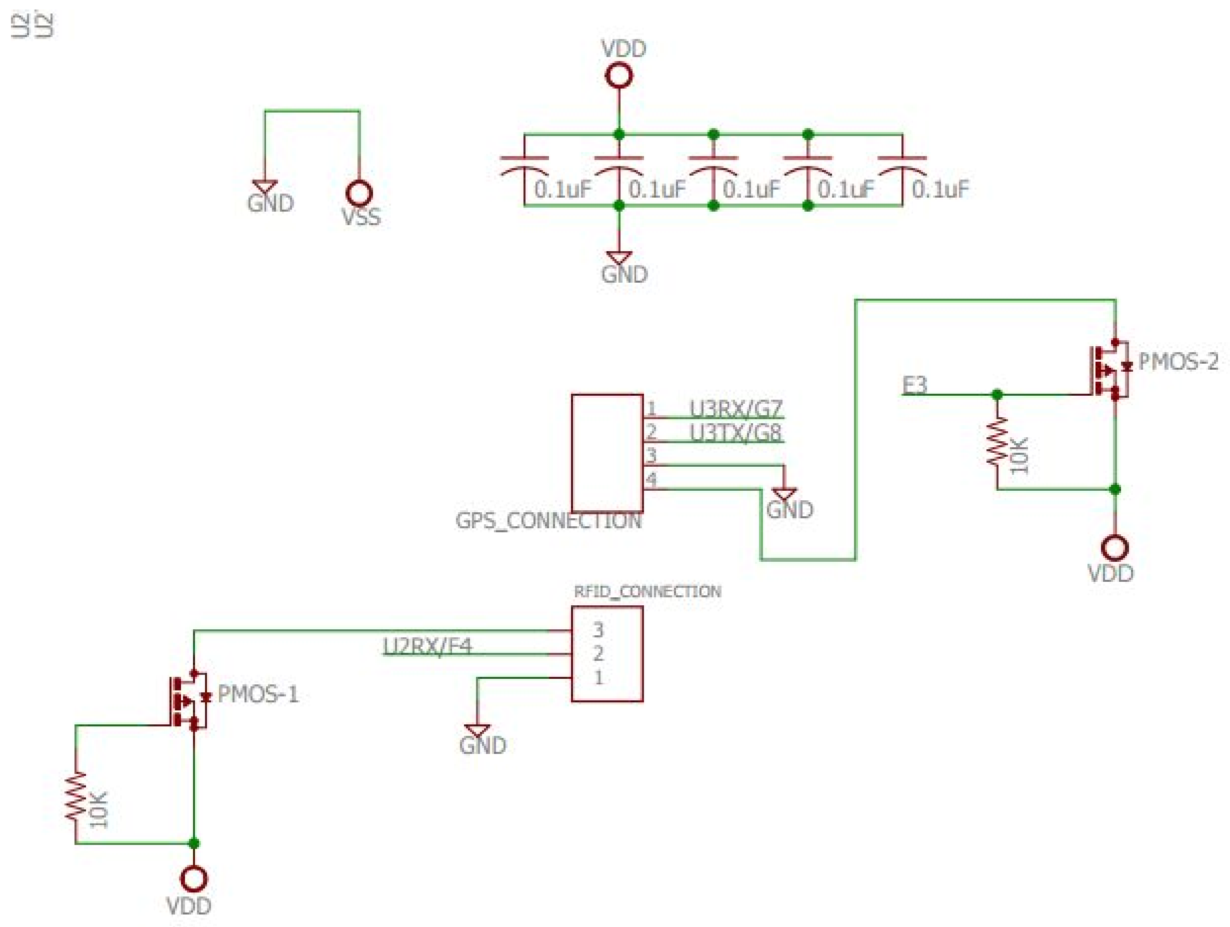
        - To limit/eliminate AC noise
    - PMOS Switches
        - To allow the microcontroller to control power to peripherals
    - PCB
        - To house all the designed electrical components of our project
    - Debugging pins
        - For debugging our PIC program using printf() statements
    - 10 Pin Block
        - For debugging our PIC program using a Logic Analyzer
- Requirements:
    - Decoupling Capacitors
        - Five .1 uF capacitors between Vdd and GN
        - Placed near the Microcontroller power pins
    - PMOS Switches
        - Three switching circuits to control RFID, GPS and WiFi power
        - Must be controllable via 3.3 V logic
        - Must be able to pass all required current (~170 mA max)
    - Main PCB
        - Must be made small enough to fit conveniently into the backpack
        - Must be designed correctly/intelligently to make the whole project work
    - Debugging pins
        - Must provide a connection to allow us to connect a computer's serial com port to one of our UART modules (UART1)
    - 10 Pin Block
        - Must provide a pin for each of our employed UART RX and TX pins
        - Must provide a pin for Vdd and GND
- Inputs:
    - Decoupling Capacitors
        - Vdd and GND
    - PMOS Switches
        - Microcontroller pin E2 to the gate of the RFID power switch
        - Microcontroller pin E3 to the gate of the GPS power switch
        - Microcontroller pin E4 to the gate of the WiFi power switch
        - Vdd
    - Main PCB
        - Battery connector
        - PICKit3 connector (for programming Microcontroller)
        - 3 pin connector (for programming WiFi chip)

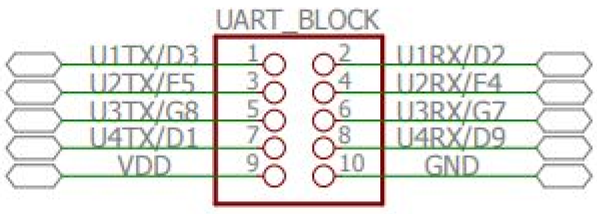
    - Debugging pins
        - Communication from microcontroller via UART1
    - 10 Pin Block
        - Communication from microcontroller via UART modules
- Outputs:
    - Decoupling Capacitors
        - N/A
    - PMOS Switches
        - Switched 3.3 V power
    - Main PCB
        - Power LED and Charging Status LEDs
        - Connections to GPS and RFID boards
    - Debugging pins
        - Output UART1 communication to computer serial com
    - 10 Pin Block
        - Output UART communication signals to a Logic Analyzer
- Why this part was chosen:
    - Decoupling Capacitors
        - It was recommended by the datasheet and confirmed by our experience with designed a PCB around a microcontroller that we should use decoupling capacitors.
    - PMOS Switches
        - The PMOS switching circuit is extremely simple in that it only require a PMOS and a current limiting resistor. Thus, it seemed like the easiest way to control power flow on the board.
    - Main PCB
        - It was a requirement of the Senior Design project to incorporate a custom designed PCB. Additionally, it was the only sensible way to connect all of our required components onto one package.
    - Debugging pins
        - We needed to have a way of getting feedback from our microcontroller in order to debug our program. The ability to print messages to the computer’s serial ports at arbitrary points in our program was an invaluable debugging tool.
    - 10 Pin Block
        - As another debugging tool, we needed to be able to monitor our UART communications so that we could debug our programs. Therefore, we designed our board to output those signals to pins

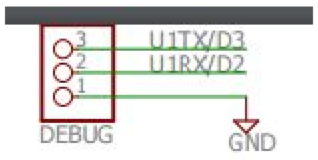
that could be measured by a Logic Analyzer, which we had available to us as a debugging tool.

- Communication protocol/programming:
  - Decoupling Capacitors
    - N/A
  - PMOS Switches
    - Controlled by the microcontroller's programming via output pins
  - Main PCB
    - N/A
  - Debugging pins
    - Laid out UART1 signals
  - 10 Pin Block
    - Laid out UART1, UART2, UART3, UART4 signals

**Schematic of Misc. Modules:**







- The top schematic shows a few of these miscellaneous components. On the top, we see the decoupling capacitors wired between Vdd and GND. Below them, we see two out of the three PMOS switches. The operation for the switches was the same in all three cases. Vdd was wired to the source, and then a current limiting resistors was placed between Vdd and the gate. The gate was also wired to the appropriate output pin on the microcontroller (you can see E3 labelled on the GPS switch). When E3 was set high, the voltage difference between the gate and source would go to zero, and the switch would act as an open, cutting off power. When E3 was set low, the voltage difference between the source and gate would be 3.3 V, the PMOS would become active, and power would flow to the drain and into the proper module. All three of the PMOS switches operated on this design.
- The middle schematic shows the 10 pin block. Vdd, GND, and all the proper UARTs are tied to a pin such that they can be monitored by a Logic Analyzer.
- The bottom schematic simply shows the debugging pins, which are connected to GND and the two UART1 signals, RX and TX.

- How this subsystem was tested.
    - The only components in this section that were specifically “tested” were the PMOS switches. We used a multimeter to verify that the switches were connected to the proper pins and working as we had designed them. This was part of our initial checking to ensure that power was getting everywhere that it needed to go. All of the other components were actually the ones that we used to test the proper operation and programming of our main components. Thus, aside of initially checking that the proper connections were made when we assembled our board, we did not have to test these auxiliary components of our design.

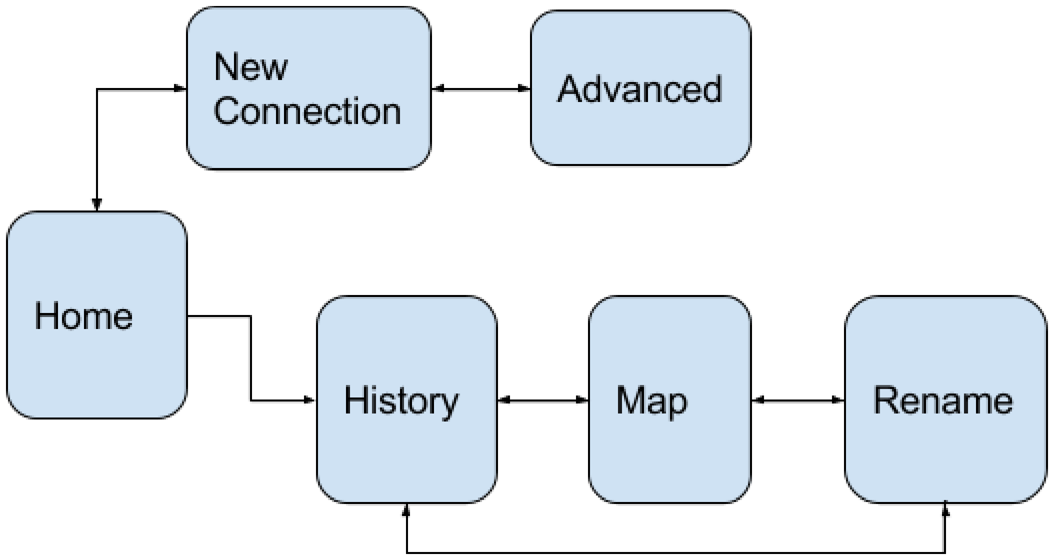
## 3.4 Detailed operation of Subsystem 2: Mobile App

### 3.4.1 Subsystem Requirements

The Mobile Application is the portal by which the TrakPak communicates all of its information to the user. Thus, there are a set of major system requirements detailing how it should ideally function. They are as follows:

- Process and relay information from backpack about items, location to user
- Easy to use interface
- Allow user to customize labels for RFID tag numbers
- Show last known location on a readable map

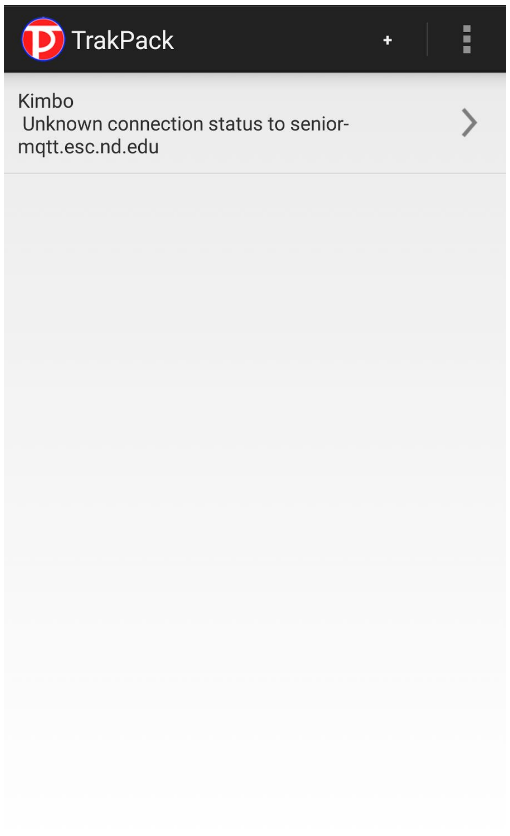
### 3.4.2 Mobile App Flowchart



### 3.4.3 Mobile App Users Manual

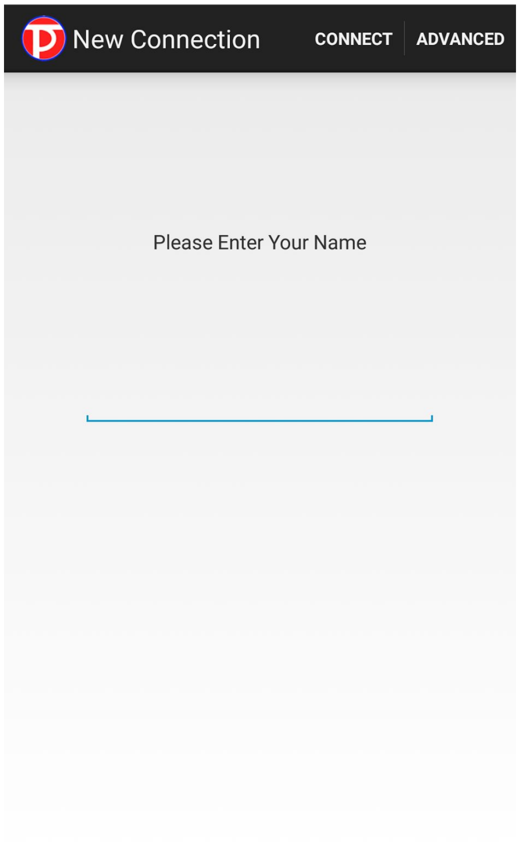
The TrakPak Mobile App has several different screens, each containing an additional way for the user to interact with the data sent by the TrakPak. The following

Users Manual is a detailed description of each page and how to best interact with the Mobile App.

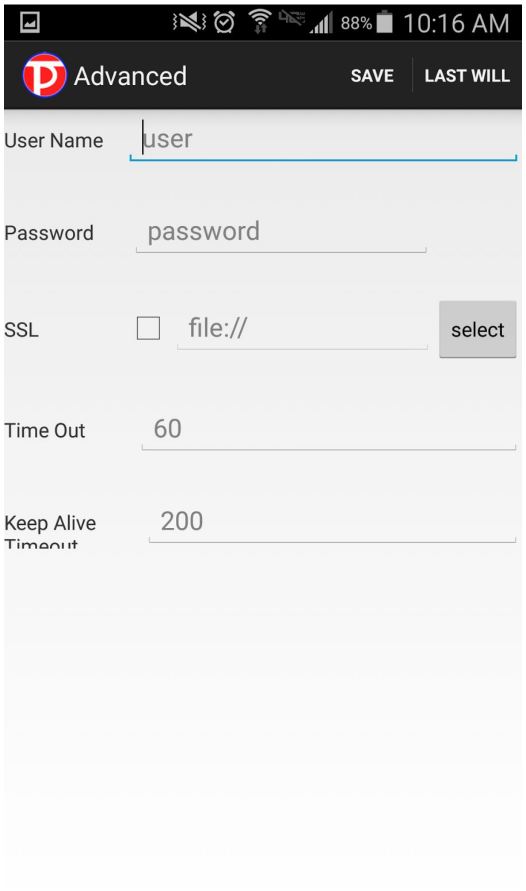


Kimbo
Unknown connection status to senior-mqtt.esc.nd.edu

This is the Home screen of the Mobile App. This is the screen you see immediately after starting the App. If you are a first time user, the white section will be blank because you haven’t created a client yet. If you are a returning user, your name will be shown as above as well as your connection status to whichever MQTT Server you are connecting to. There are two main ways to interface with this screen. The first is to click on the small “plus sign” in the top right corner. This will take you to the New Connection screen, the screen where a user can create a client. The second interface is to click on the connection name on the list. This will connect you to the server and will take you to the History screen.



This is the New Connection screen. The user needs to only enter his name and press the Connect menu button to establish a new connection. This will return the user to the Home screen. The Advanced menu button will take the user to the Advanced screen.



10:16 AM

Advanced

SAVE

LAST WILL

User Name user

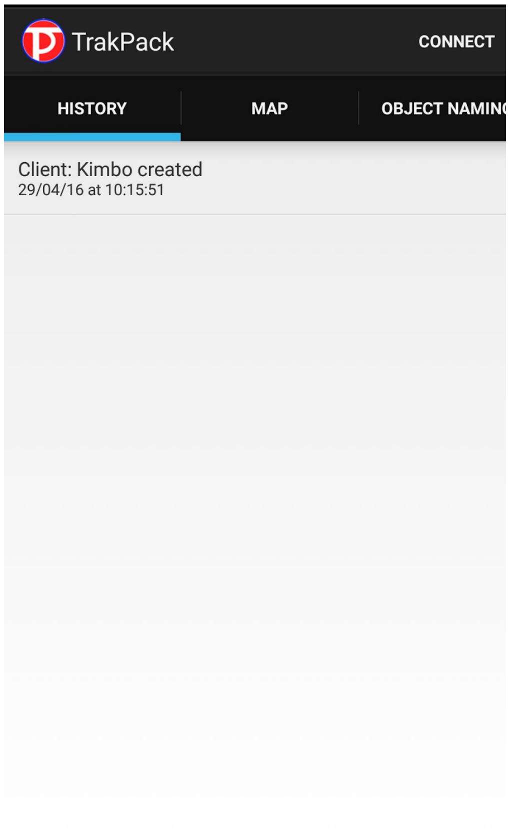
Password password

SSL ☐ file:// select

Time Out 60

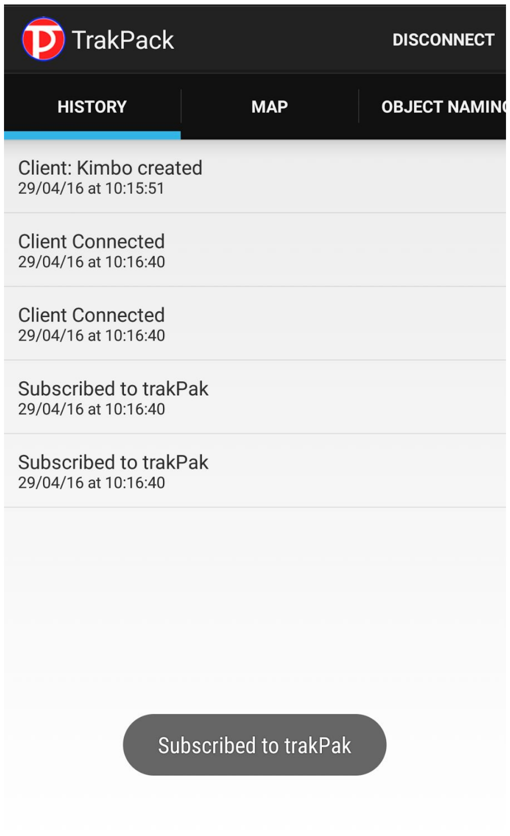
Keep Alive Timeout 200

This is the Advanced screen and should only be used in special cases.  This is to customize the Mqtt server interactions.  For the scope of this project, it is unnecessary to use this screen.  However, future improvements on this project might require such specifications in order to set users apart and keep their data private.



Client: Kimbo created
29/04/16 at 10:15:51

This is the start state of the History screen. This is the screen where every scanned item will be shown with a time stamp. It is also the screen that will notify you when you are connected to, subscribed to, or disconnected from the MQTT Server. The History, Map, and Object Naming tabs represent different screens for processing or displaying information. You can navigate between them by either clicking on the tab directly, or by swiping left or right on a given page. The “Connect” button in the top right corner is consistent for all three tabs, and is a crucial part of the app. The “Connect” button both connects you to the MQTT Server and subscribes you to the TrakPak topic. Once clicked, the History screen will look like the following:



Client: Kimbo created
29/04/16 at 10:15:51

Client Connected
29/04/16 at 10:16:40

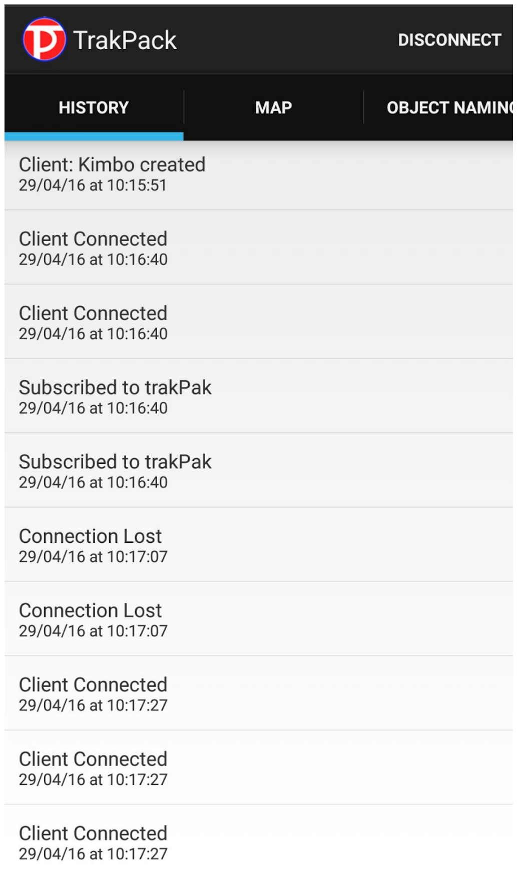
Client Connected
29/04/16 at 10:16:40

Subscribed to trakPak
29/04/16 at 10:16:40

Subscribed to trakPak
29/04/16 at 10:16:40

Subscribed to trakPak

This screen is the History screen when the Mobile App is connected to the MQTT Server. Three things are instantly noticeable: the new button, the pop-up message and the history log. Firstly, once the App is connected to the server, the “Connect” button changes to a “Disconnect” button. As the name implies, this button now actively removes you from the MQTT Server. This is especially useful when you are at home or do not have your backpack and you want to save power on your phone. The second noticeable change is the history log. Obviously, there have been a few postings informing you what has happened since you hit the “Connect” button. These are the common postings when a connection is made. The double message is a bug in the system that was never flushed out of the system, and so was never fixed. The third noticeable change is the temporary message on the bottom, showing the “Subscribed to trakPak”. This is simply an update to the user that can be seen on any screen as soon as the App is connected to the MQTT Server.



Client: Kimbo created
29/04/16 at 10:15:51

Client Connected
29/04/16 at 10:16:40

Client Connected
29/04/16 at 10:16:40

Subscribed to trakPak
29/04/16 at 10:16:40

Subscribed to trakPak
29/04/16 at 10:16:40

Connection Lost
29/04/16 at 10:17:07

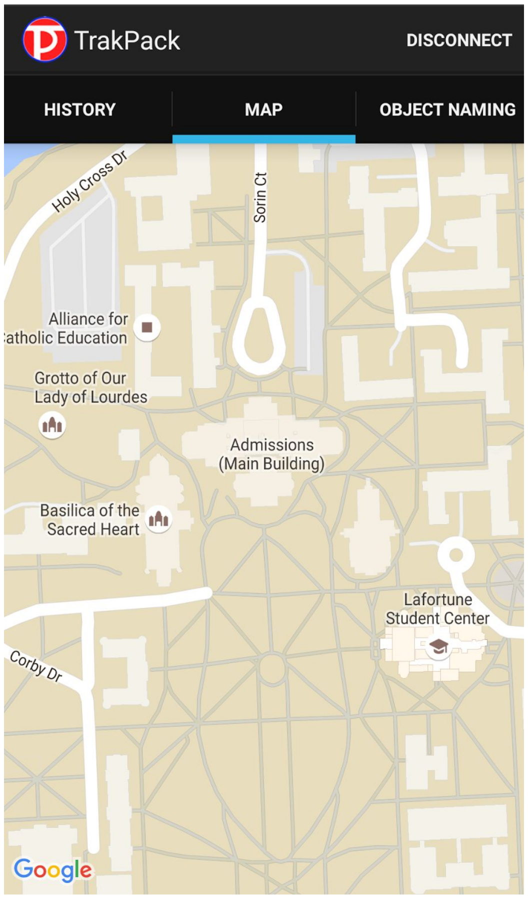
Connection Lost
29/04/16 at 10:17:07

Client Connected
29/04/16 at 10:17:27

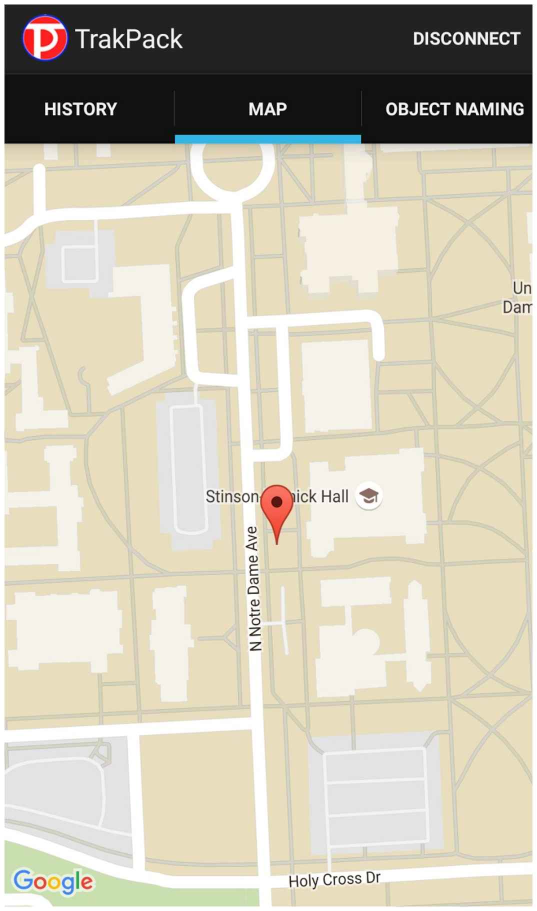
Client Connected
29/04/16 at 10:17:27

Client Connected
29/04/16 at 10:17:27

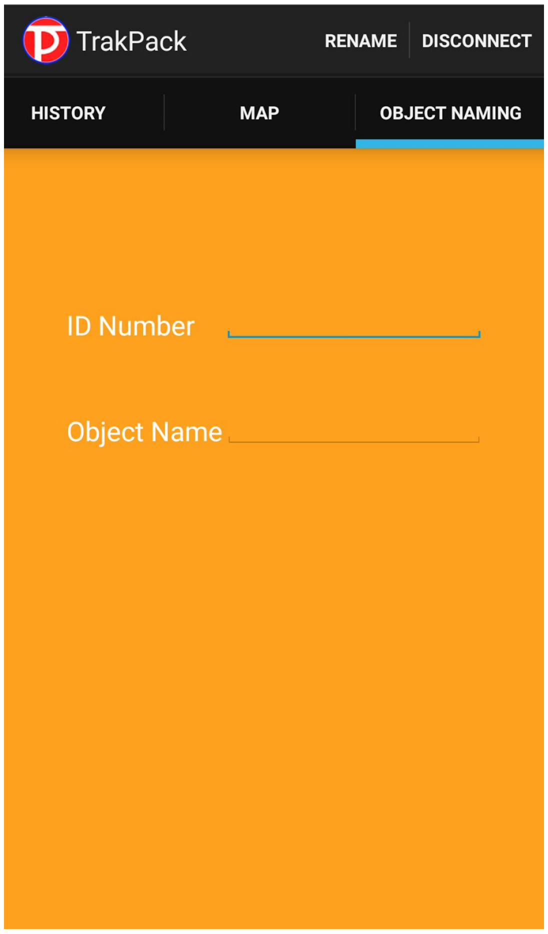
Occasionally, the Mobile App will lose connection with the MQTT Server. When this happens, the Connection Lost message will be displayed on the screen. To re-connect, simply click the “Connect” button, as the above user did, to re-connect to the Server. Besides the connection notifications, any scanned object will be displayed on this screen along with a timestamp. In clicking on that notification, the user will be placing a marker on the Map screen. Once they navigate to the Map screen, the user will be able to see where his item was last scanned.


TrakPack
DISCONNECT
HISTORY
MAP
OBJECT NAMING
Holy Cross Dr
Sorin Ct
Alliance for Catholic Education
Grotto of Our Lady of Lourdes
Admissions (Main Building)
Basilica of the Sacred Heart
Lafortune Student Center
Corby Dr
Google

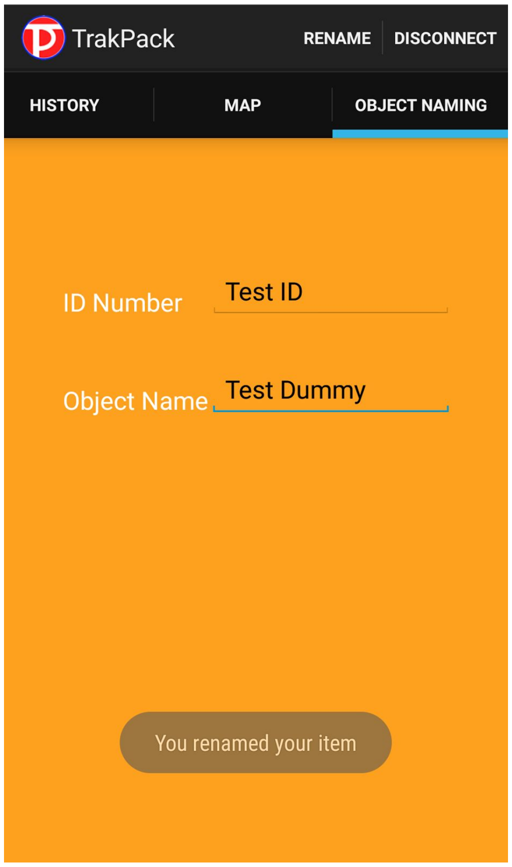
This is the Map screen. The Map screen defaults to the above location when no marker has been set yet. Once the user has selected a scanned object on the History, the map will look similarly to the image below. The map is fully scrollable and features all the features of a typical Google Map interface.



This is what the Map screen looks like when it has a marker placed on it. If the marker is clicked, then the item name pops up. In the bottom right corner of the display there is an icon that would allow a user to take the gps location of the item and access Google Maps in order to get directions to the item.



This is the Object Naming screen. Once an object has been scanned for the first time, it shows as a number instead of an item name. If that number is put in the ID Number line, then the user has to merely type a relevant item name on the Object Name line and press the Rename button on the menu bar. If that item is then scanned again, then it will appear as whatever the user named it. The screen below shows the toast when an item is successfully renamed.



ID Number Test ID

Object Name Test Dummy

You renamed your item

### 3.4.4 Mobile App Testing

The Mobile App was tested mostly by sending a message to the topic the App was subscribed to via a computer program and following that input through every screen of the App. This allowed first the testing of the Mqtt functionality, and then the testing of functionality for the rest of the App. The majority of the Mqtt functionality is easily tested for, on account of that being the only real input to the App. After that input is established, there was a lot of testing done to ensure that the three fragments (History, Map, and Object Naming) communicated together properly. This testing was done by focussing on one process at a time and stepping through the code to ensure robust coding practices. Then we attempted to test the code with every possible instance that it would see in order to check for quality and robustness.

## 3.5 Interfaces

All interfaces between the modules have been sufficiently discussed in Sections 2 and 3 above.

# 4 System Integration Testing

## 4.1 Describe how the integrated set of subsystems was tested.

The first task for testing our integrated system was confirming that our board design and assembly process had produced a board that had all the proper components connected in all the proper places. In some sense, this was Step 0. Before we could even worry about making our project functional, we had to confirm that the hardware was properly connected. As mentioned above in the subsystem testing procedure, we checked to make sure that Vdd and GND were not shorted. We checked all the pins of our small packaged parts to confirm that there were no shorts or missing connections, especially on the power pins. This process was a bit tedious, but we caught a few mistakes while they were still small and fixed them before they became large problems.

Once the system was fully “integrated” (all subsystems were properly connected and talking to each other, everything was properly powered, etc.), the testing procedure took place in two major steps.

Step 1 - Indoor Testing (Code Debugging)

When we got the point where we had assembled our own board and confirmed that everything was connected as it should be, our base code was probably about 80% functional in terms of satisfying our requirements. Thus, the first task was to finish writing our main program for the PIC. This process was done by writing a program, running it on our board, and then monitoring the activity via the UART1 serial port, Logic Analyzer and multimeter use. This process was a bit tedious, but after many iterations we finally got the code to a functional state.

From there, we spent some time indoors with our system trying to break it. That is, we basically tried to think of ways that a user would use the TrakPak that our current code could not properly account for or handle. Based on those actions, we had to modify our code and then test that modified code. This process took a little while as well, but it was important because it allowed us to produce a much more robust system before ever even taking it outside. It was during this indoor testing that we added the feature of saving scan information if there was no WiFi in range, and having the GPS save its last known location in the event that we cannot get a GPS fix. During this time, we debugged the phone app in much the same way. Without going outside at all, we were able to test the functionality in the app.

Step 2 - Outdoor Testing (Code Refining)

After fully testing the system as much as we could inside, we then started to run outdoor tests. These tests were not as in depth as the previous testing because we only wanted to ensure that we had a functional prototype, so we only needed to confirm that our TrakPak was functional per our requirements. However, this testing was still vital. We were able to catch a few more bugs and update the code appropriately to fix them. We also were able to learn that our packaging and orientation did not seem to affect the WiFi and GPS modules' abilities to get a good signal. Nevertheless, we would want to take care to make sure that we do not do anything to interfere with or block these signals. The final testing that took place was a walk around campus with the backpack on and the phone app in hand. We walked from Fisher Hall to Debartolo, and then from Debartolo down to Eddy Street. From there, we returned to Stinson-Remick. All the while, we scanned items in and out and checked to make sure that we were seeing the proper updates on the app as appropriate. With this test being successful, we ended our testing of the prototype.

## 4.2 Show how the testing demonstrates that the overall system meets the design requirements

To show that our testing satisfied our requirements, we have repeated our requirements below, and then with each requirement, we commented in bold how our testing showed that we satisfied the requirement.

**Embedded System Requirements**

- PIC Microcontroller
    - All required passive components (Vcap, Decoupling Capacitors, etc.)
        - **All required components were designed and assembled on the board**
    - Programming Requirements
        - Language
            - C
        - Environment

            - MPLAB IDE
        - Hardware
            - PICKit 3
            - Pins on board for PICKit 3
        - **All coding and programming was done in accordance with these requirements.**
- General Requirements
    - Must be able to communicate with all auxiliary hardware, including but not limited to:
        - GPS chip
            - **UART3 communication was verified and successful**
            - **Successfully commanded the GPS module**
            - **Successfully read back GPS data**
        - Power Hardware
            - **All PMOS switches worked as designed. Power was controlled to the other modules**
            - **The microcontroller itself was successfully powered**
        - WiFi chip
            - **UART4 communication was verified and successful**
            - **Data was successfully sent to the WiFi module**
            - **Acknowledgements were received from WiFi module**
        - RFID Scanner
            - **UART2 communication was verified and successful**
            - **RFID tag numbers were read back into microcontroller**
    - Must have at least 4 available interfaces for proper serial communication with peripherals
        - Each auxiliary hardware piece must have an available communication protocol (UART, SPI, I2C, etc.)
            - **4 UART modules worked successfully**
    - Must have at least 3 I/O pins for controlling power
        - **Digital output pins E2, E3, and E4 worked**

### 2.2 Power Requirements

- General Requirements
    - Input: Must be rechargeable via USB connection or connection to wall
        - **Ability to charge battery was verified by testing**

- Output: Must be able to connect to our microcontroller via a microUSB connection
  - **We modified this requirement thanks to an external USB-UART device provided by the professor. Successully communicated with UART1 for debugging, confirmed by testing.**
- Weight Requirement
  - As light as possible considering that the user will have to carry it around
  - Maximum: 5 lbs
    - **Easily came in under 2 lbs with all of our circuitry, not juut the battery**
- Size Requirement
  - Shape of battery must not be bulky such that integration into the backpack is difficult and/or the backpack is not as usable
    - **Battery was flat and easily mounted in the same box as our PCB**
- Total Maximum Power Draw: ~500 mA
  - The above power estimate has been made by choosing a likely candidate for each component and then determining its maximum power draw. This should give us a worst-case scenario power estimate.
    - Microcontroller
      - PIC Datasheet
      - Maximum Input Current = 300 mA
    - GPS chip
      - Gtop GPS
      - 30 uA (standby current) at 3.3V = 54 uW
    - LED Hardware
      - Average Diode Forward Current = 150 mA
      - Depends on current limiting resistor choice
    - WiFi chip
      - ESP8266
      - 15 mA typical sleep value, 170 mA TX max, 50 mA RX max
    - RFID Scanner
      - ID-12LA Datasheet
      - 35 mA maximum
- Required Runtime: 12 hours
- Required Battery Estimate: 7200 mAh

        - **This estimate was the absolute maximum for the worst-case, most poorly engineered design. Thus, we selected a 5000 mAh battery, which still was far above our needs.**
        - **The power solution was able to source enough current at all times based on our testing, and all devices received proper power and functioned as designed.**

## 2.3 Wireless Interface Requirements

- General Requirements
    - Must be able to communicate with WiFi chip connected to the central PIC board and RFID scanning device
        - **UART4 provided the proper communication channel between the WiFi chip and the PIC**
        - **Received acknowledgement from the WiFi**
        - **Successfully sent data to the WiFi chip**
        - Must be able to connect to any unsecured, public networks
            - **We did not implement this requirement, but it turned out that for our prototype, we only had to be able to connect to ND-guest to achieve the functionality that we wanted, which was a WiFi connection throughout the majority of Notre Dame's campus**
    - Phone system and WiFi chip both must be able to communicate within the domain of the created mobile app
        - **Successfully sent information from the WiFi chip to the phone app via the class  Amazon MQTT server**
        - **Phone App successfully could pull, parse, and present data from the MQTT server**

## 2.4 User Interface Requirements

- General Requirements
    - Must be able to alert the user that each item has been successfully or unsuccessfully scanned
        - This will be done with a beep on the RFID board
        - **Successfully implemented a non-intrusive beeping sound using the Sparkfun RFID chip**

    - Must have a functioning mobile app to provide scanned information for each item (Object, GPS & Time Stamp)
        - **The phone app provided a history of the object that was scanned and when it was scanned. If the user tapped that object, they could navigate to a map that showed a pin with that object's location.**

### 2.5 Installation/Use Requirements

- General Requirements
    - The TrakPak will not be user-installed; we will install and assemble all components.
        - **This did not change; the TrakPak is still installed on our end.**
    - The passive RFID tags will be installed by the user onto belongings of their choice via stickers.
        - **We found cheap, non-obtrusive RFID stickers that stuck onto most everyday objects that we tested.**
    - The following components should be able to be installed directly on or connect directly to a PCB of dimensions ~4 in x ~3 in.
        - Microcontroller (and all its required passive components)
        - Clock Circuitry
        - GPS Chip
        - WiFi Chip
        - **The microcontroller and WiFi chip were successfully installed on a PCB of dimensions 3" x 3.5", the same size as our battery. The GPS chip was installed on its own PCB board of about 1.75" x 2" so that it could be mounted closer to the top of the backpack. We did not require clock circuitry by the end of the project, because we instead used the GPS data for our time.**
    - The RFID scanner must be installed in an easy-to-reach area of the interior of the backpack to protect it from weather.
        - **The RFID scanner was installed on the inside of the backpack, close to the top, such that it was both easy to access and protected from the elements.**
    - The battery and microchip must be installed in a convenient location within the backpack to prevent damage to the battery and preserve interior space.

        - **The battery and main PCB board were housed in a Polycase plastic case, which protected them from damage and unnecessary motion. This case was mounted in one of the top pockets of the backpack for our prototype.**
        - The battery's power cord must have a way to be secured when not in use.
            - **The USB cord could be wrapped up into this same top pocket if the TrakPak was not being charged.**
    - All wiring must be secured and well hidden to prevent damage from wear and tear as well as preserve a sleek look.
        - **All wiring and boards were secured in Polycases or tucked away in a pocket, such that the user would not see any electronics aside from the RFID scanner.**

## 2.6 Safety Requirements

- Must ensure that the battery can account for overcharging without overheating too much.
    - **The battery charging circuit had built-in charge limiting to avoid over-charging the battery.**
- Ensure that the battery charging process is well protected to avoid delivering shocks to the user.
    - Only possible dangerously high voltage is from charging the backpack off the wall.
    - **All electronics were tucked away from the user and away from the elements, such that normal use would not present the user with any risk of shock.**
- As a backpack, do not overburden yourself to avoid damage to spinal system.
    - **Our backpack is lightweight, and our electronic additions are not perceptible as additional weight.**

## 2.7 Mechanical Requirements

Our project is designed to improve a user's experience with a backpack. As such, there are several main mechanical requirements that cannot be exceeded or the TrakPak would cease to be an advantageous product. Primarily, the TrakPak must be able to be fully functional inside of a backpack without compromising the carrying

capabilities of a comparatively sized backpack.  In order to meet this expectation, we will have to use hardware that is small, lightweight, and durable.
**Our backpack was of typical size, lightweight even with all of our electronics installed, and the electronics were durable enough, considering that it was a prototype.**

The first consideration is size.  Our hardware will need to be small enough to be incorporated into an existing backpack without taking up too much space.  The TrakPak would be useless if there was no room to put anything into it.  Therefore, we will need to buy parts that will minimize space while keeping effectiveness high.  The major challenge for this consideration will be the power supply, since most power supplies are relatively large compared to the average microcontroller.  The specific size requirement will be determined by the backpack that we choose to augment, however as a general rule we will need to buy the smallest parts possible that fit our requirements.
**We passed our size requirement, in that our battery and board (the largest components) fit snugly into a small plastic case and easily tucked out of sight within the backpack.**

The second consideration is weight.  Backpacks quickly become uncomfortable when they are filled with too much weight.  While we may not be able to decrease how much a user tends to put inside the TrakPak, we can control how much the base weight is for an empty TrakPak.  In order to keep this base TrakPak weight at a reasonable level, we will need to find parts that have minimal weight.  Most of our components, like the microcontroller or the LEDs, will be of negligible weight.  However, the power supply may be another challenge as batteries with the power capacity that we will need tend to be comparatively heavy.  The specific base weight requirement for the TrakPak will be highly dependent on the weight of the backpack we choose to augment, however our other components should be under five pounds altogether.
**All electronic components and cases combined barely amounted to 2.5 pounds, so we passed our weight requirement.**

The third consideration is durability.  The TrakPak would be a highly ineffective product if it couldn't stand up to the rigors of daily life.  Few backpack users treat their backpacks with special care.  They are often dropped on the ground, stepped on, or in other ways misused.  We cannot assume that the TrakPak will be treated any differently than a regular backpack by a user, and so our components will need to be able to survive these daily abuses.  There are two ways we can meet the durability requirements.  The first is by purchasing already durable parts.  Some parts, like the microcontroller and its immediate plugins will be inherently fragile, but other parts, like

the power supply or the RFID scanner, may have variable durability.  The second way to meet our durability requirement is by packaging our parts.  This packaging will need to meet the size and weight considerations, as well as sufficiently protect our hardware from the daily wear and tear of backpack usage.
**Our prototype was durable enough that the backpack functioned properly as we tested how a typical student would use it. While it probably would not stand up to the rigors of a whole semester of use, it was durable enough to function as a working prototype.**

In conclusion, the TrakPak is dependent on three mechanical requirements. Firstly, the hardware must be small enough to not hinder the TrakPak's carrying capacity.  Secondly, the hardware must be light enough to not inhibit the user's comfort. Thirdly, the hardware must be protected so that the TrakPak's technological components remain functional.  These three mechanical requirements will ensure the longevity of the TrakPak functionality and the user's satisfaction with the physical construction of the TrakPak.

# 5 Users Manual/Installation manual

## 5.1 How to install your product

In order to install this product, the user must buy the TrakPak, and download the arduino firmware that is included after the product is purchased. All the user must do is open the file up, and specify what WiFi networks will be primarily used by the user. This information should be included in the NetID and Password fields. By inputting multiple WiFi specifications the user will be able to utilize the TrakPak on more networks. After the WiFi networks are set up, all one needs to do is click program, and put the TrakPak back in it's packaging so that it can be installed in the backpack. The enclosed TrakPak box should be placed inside the bag with the RFID scanner connected as well. The next step to installation is to place the circular adhesive RFID tags on items that one wants to track in the TrakPak. Once all these steps are completed one should download the TrakPak mobile app to their smartphone so the installation can be fully complete.

## 5.2 How to setup your product

In order to set up the TrakPak once everything in the backpack is enclosed, the user should place the RFID scanner in the interior of the backpack where the user is able to easily and conveniently scan items in and out of the backpack. This spot will differ for every user, so this is an important step to the setup.

It is also important to do some preliminary scans of the RFID tags in order to label them in the app prior to everyday use. This is the place where you can scan a tag and name it things such as calculator, notebook, or water bottle. Finally one must make sure the battery is charged before using. It can be charged using a USB port, and it is recommended that it is charged in the evening. Hopefully this is an effortless last step before the user can go out and put the TrakPak to daily use.

## 5.3 How the user can tell if the product is working

Once the TrakPak is on, items have been tagged, WiFi networks have been specified, and the mobile app has been downloaded, the user is able to let the TrakPak do the heavy lifting. Of course it is important for the user to be able to tell if it working. This is actually a very straightforward test. Simply take an item out of the backpack and check the update on the mobile app to see if it gives the proper GPS and time location. If it does you know the TrakPak is working and if not, you know you got one of the very very few malfunctioning TrakPak's.

Some user's may be confused if they hear a beep when they scan, and other times not a hear a beep. This is something the user should take note of. If they try to scan an item and the RFID scanner does not beep, they must scan that item again. The reason it did not scan is because the TrakPak is busy thinking and sending the information of the previous scan. Once the previous scan is sent and the TrakPak is ready for another input, the RFID scanner will be enabled once again so the user can scan more items.

## 5.4 How the user can troubleshoot the product

If for some strange reason the user is experiencing difficulty in using the TrakPak, or it malfunctions, there are a few ways the user can troubleshoot their product before sending it into the experts. The most common issue we believe users will face is the RFID scanner not turning on after a timeout. This may occur due to low battery typically, or a strange state when the firmware gets stuck in a gps loop. In order to fix this issue, the user should open up the TrakPak box, and click the reset button. This should solve the problem and allow the user to go right back to where they left off. It would also be a good idea to charge the battery if this occurs.

Other than this issue of the RFID scanner sometimes not turning on, the user should not experience issues. If the user does end up experiencing other issues they must either consult a TrakPak expert or have tremendous MPLAB programming skills to try to debug the issues personally. The user should be able to just restart the app if it ends up crashing for some reason.

# 6 To-Market Design Changes

The first generation TrakPak was a great prototype for sure, but had our team had more time and budget we think the product could have been taken to the next level. Highlighted below are a few key ways we think our product could have been improved, and some design targets for the second generation TrakPak:

- Include an accelerometer on our design to be smarter about triggering when to receive GPS data. This would essentially keep the GPS on until the accelerometer sensed that the user was moving. Once the microcontroller is notified by the accelerometer that the user is moving it will begin to request data from the GPS. This was the TrakPak is only using the GPS when it needs to: when the user is moving to a new place. This would improve battery life and efficiency of our program for when this product is on the market.

- Although it is doable, a user isn't going to want to scan in items every time they put them in and out of a backpack. In order to improve this we are looking into incorporating low energy bluetooth into our backpack instead of RFID Scanning. This will allow users to tag their items with a bluetooth beacon (such as an iBeacon) and then have a bluetooth reader in the bag so it can tell which beacons are in the bag due to proximity instead of having to always scan. This feature would be very nice, but would cause a price increase in the TrakPak because the beacons are much more expensive than adhesive RFID tags.

- The TrakPak is a great backpack, but we would like "TrakPak" to be more versatile than just one backpack. If we were able to get the guts of the TrakPak packaged in a neat and tight portable package, we would be able to sell the TrakPak as a device that anyone can put in any backpack, purse, etc. By doing this the uses of our product would increase, and people could put it in whatever backpack they want, making it more marketable. Another perk is people wouldn't have to buy new backpacks if they wanted the TrakPak.

- Though the WiFi form of the TrakPak is very practical to college campuses or places where WiFi is heavily prevalent, it may not be the most practical where that isn't the case. To improve on this issue we would like to transition the TrakPak from a WiFi device to a cellular device so that one can get all the perks of the TrakPak anywhere they have cellular service. This may make the TrakPak come with a small rate plan, but if we team up with cellular companies this could simply be included on somebody's phone plan. This would allow much more versatility and expand where the TrakPak can be a valuable asset.

- A final improvement that is much needed before this product is taken to market is the ability to turn the TrakPak on/off with a switch as well as a way to see how much charge is left on the battery. This is an absolute must for the product version because users need to know when they need to charge the TrakPak and they need to be able to turn it off if they don't want to use it for a period of time. This would be a feature so the users can have more control of when they want to use the TrakPak as well as giving them the ability to see when they need to charge it up so it doesn't die while in use.

# 7 Conclusions

This project introduced a wide scope of necessary applications that pulled experience from many of our learning experiences as students of Notre Dame's electrical engineering program. While there weren't any occasions for our specific project to use knowledge gained from classes involving signals and systems, we still utilized problem solving abilities gained from classes involving coding, embedded systems and electronics. We had to step up our abilities in terms of software in order to understand how mobile applications work, yet we got everything we planned to get done completed and integrated into the final product.

As stated in the beginning of this paper, we are extremely pleased with how our product has turned out and fully endorse its use as a functional prototype that has a strong case in terms of its patentability. Our design is aesthetically pleasing and gets the job done. The hardware is all fully functional and is able to scan in an RFID, give the user an auditory cue for a successful scan, gather GPS data, and communicate all information via WiFi to an MQTT server that can communicate with our Mobile Application. While the GPS cannot give extremely accurate information while indoors, it is fully capable of providing the user with good enough of a guess as to where an item might be. The app is fully functional to the point where a person has to manually enter the serial data unique to each RFID tag, and is able to keep a running history of each item and the of the backpack itself. It is able to visualize the location of each item via Google Maps. The weight of the design overall adds next to nothing in terms of any sort of noticeable weight, depending on the size of the battery that a person chooses to include. In all, we conclude that we have created a working prototype that can help college students cut down on time looking for items that should always travel with them. We are confident that this design has copious implications for further development and can be applied to many different lifestyles, not just those that are found on university campuses. We believe that the concept can be extended to a wide variety of activities ranging from corporate work in the city to adventurous hikes in the mountains. If we were to have more time to gain knowledge and work on further hardware development, we could add this next level of functionality. While the project is far from fulfilling its ultimate potential, we are on the right track, so to speak, and have many ideas that we are confident would help a product like this thrive in the marketplace.

We fully support the use of the TrakPak, regardless of whether you decide that the technology is to be used in a backpack or in another implementation. The aid in memory is invaluable and far and wide can assist not only those who suffer from occasional lapses in memory, but even more so those with diagnosed Parkinson's and Alzheimer's disease. The technology itself can exist in any environment where tracking items is necessary and forgetting items is possible. We believe that we have started the TrakPak down its path as a simple but driving force for good.

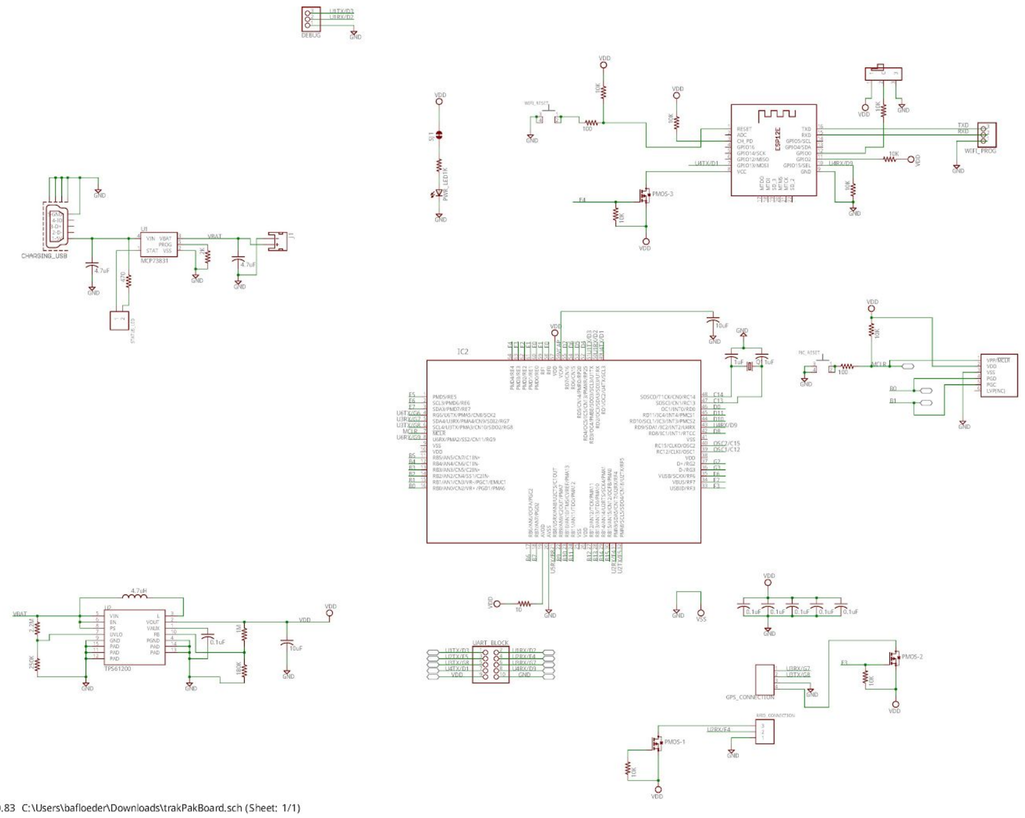
# 8 Appendices

Complete hardware schematics
Complete Software listings
Relevant parts or component data sheets (do NOT include the data sheets for the microcontroller or other huge files but give good links to where they may be found.)

## 8.1 Hardware Schematics

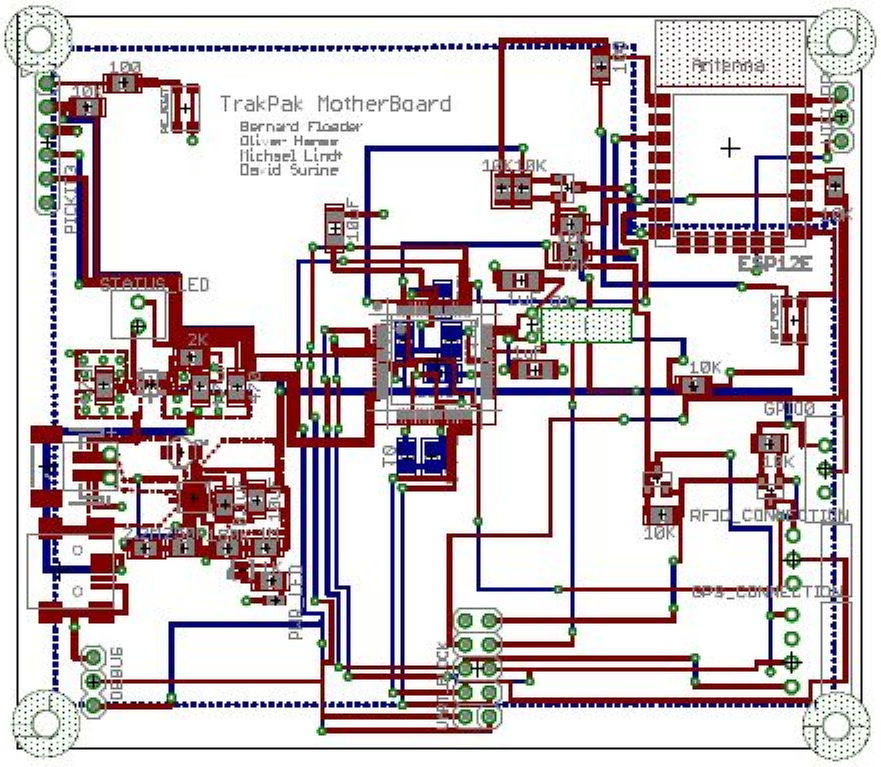
### 8.1.1 Main Board (Schematics and Board)

## Schematics:



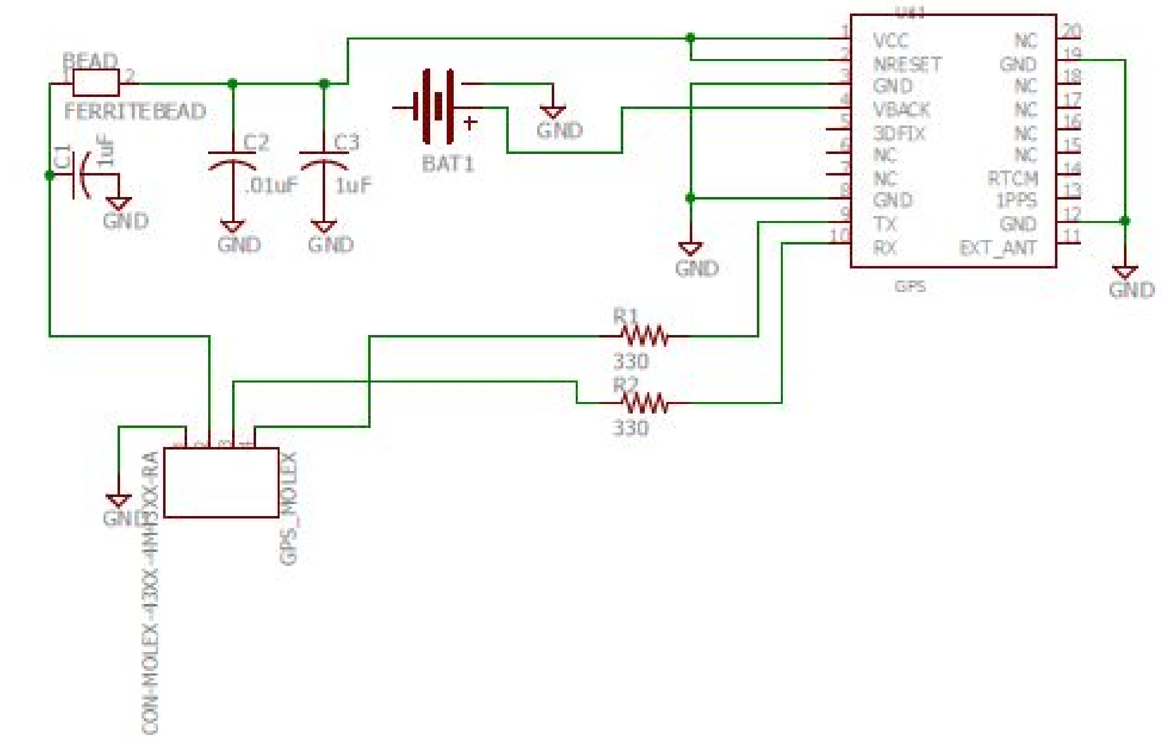
4/27/2016 3:47 PM f=0.83 C:\Users\bafloeder\Downloads\trakPakBoard.sch (Sheet: 1/1)

**Board:**

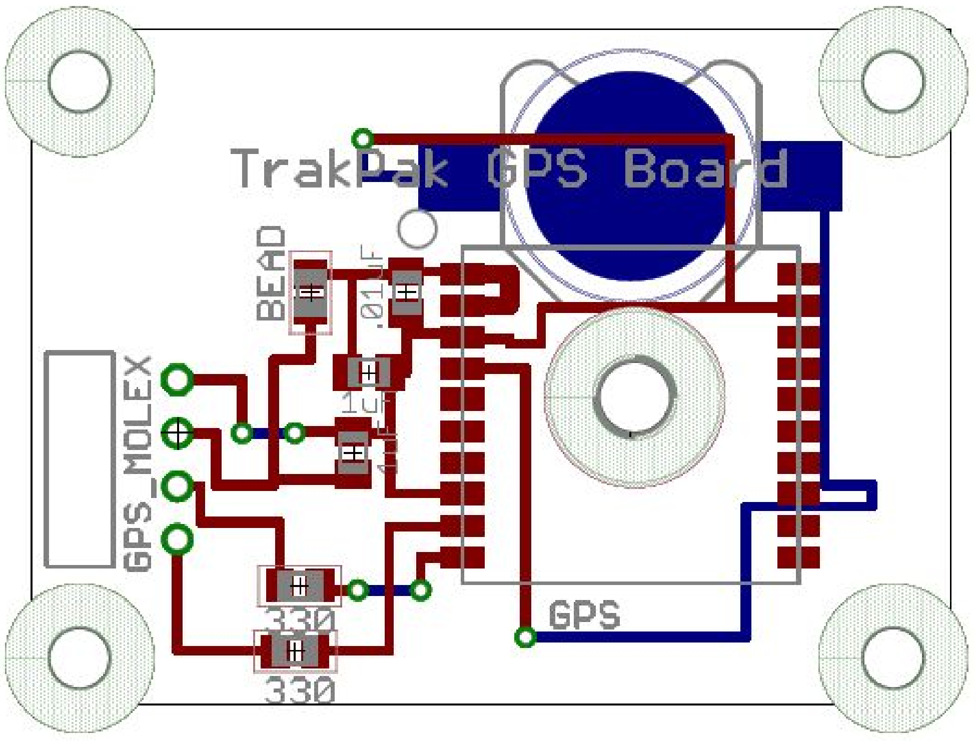

TrakPak MotherBoard
Bernard Floeder
Oliver Hamer
Michael Lindt
David Surine
Antenna
ESP12E
STATUS_LED
PICKIT3
DEBUG
GPIO0
RFID_CONNECTION
GPS_CONNECTION
PWR_LED
10K
2K
100

## 8.1.2 Gps Board (Schematics and Board)

**Schematics:**

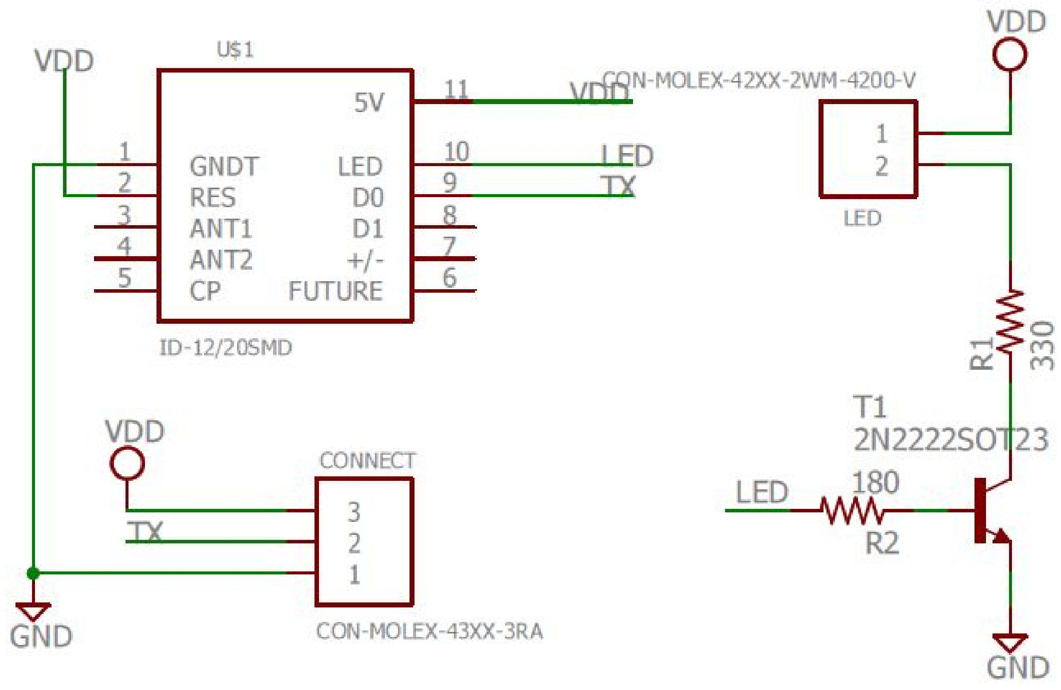


**Board:**

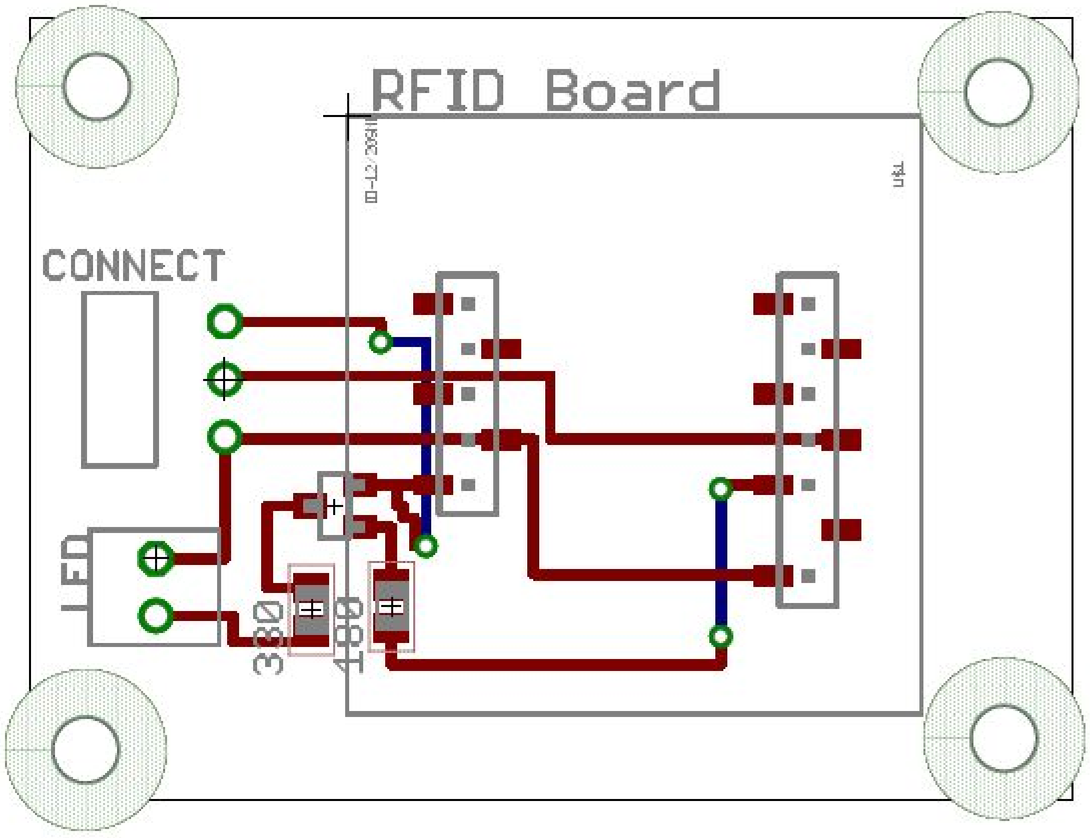


## 8.1.3 RFID Board (Schematics and Board)

**Schematic:**



**Board:**



## 8.2 Software Listings

### 8.2.1 Hardware Code

**Arduino Code:**

```
#include <ESP8266WiFi.h>
#include <MQTT.h>
#include <PubSubClient.h>
#include <SoftwareSerial.h>
/************************* WiFi Access Point *********************************/

#define WLAN_SSID       "ND-guest"
#define WLAN_PASS       "jimmyeatworld"

/************************* Adafruit.io Setup *********************************/

#define SERVER_ADDRESS      "senior-mqtt.esc.nd.edu"  // server in 213 SR

#define SERVER_PORT        1883       // standard port
```

```
String str;
int i;
int j;

// Create an ESP8266 WiFiClient class to connect to the MQTT server.
WiFiClient wf_client; // instantiate wifi client
PubSubClient client(wf_client, SERVER_ADDRESS); // pass to pubsub


void setup() {
// Setup console
  Serial.begin(9600);
  Serial.swap();
  delay(10);

   // Connect to WiFi access point.
  i=0;
  WiFi.begin(WLAN_SSID, WLAN_PASS);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
      //Serial.print(".");
      i= i + 1;
        if (i == 40)
        {
          Serial.print("s");
        }
    }
   if (WiFi.status() == WL_CONNECTED){
    Serial.print("g");
   }
}


void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    delay(10);
      if (client.connect("mydevice")) {

            if(Serial.available() > 0)
              {
                  str = Serial.readStringUntil('+');
                  Serial.print(str);
              }
      if (str == ""){}
      else{
      client.publish("trakPak",str);
      delay(50);
      Serial.print("z");
      str = "";
      }

     }


  }
}
```

**Mplab Code:**

```
/*
 * File:   rtcc1.c
 * Author: dsurine
 *
 * Created on April 11, 2016, 3:18 PM
 */

// Working on the final code here
// GPS and RFID code, no WiFi yet

#include <stdio.h>
#include <stdlib.h>
#include <proc/p32mx695f512h.h>
#include <xc.h>
#include "configbits-16ex8.h"
#include "SDlib16.h"

#include <sys/attribs.h>

void UART_init(void){

    // Set up UART1 = USB to computer

    U1MODEbits.ON = 1;
    U1MODEbits.BRGH = 0;
    U1STAbits.URXEN = 1;
    U1STAbits.UTXEN = 1;
    U1BRG=51; // Setting baud rate to 9600 manually (BRGH=0)

    set_output_device(1);

    // Set up UART2 = RFID (Rx is RF4, Tx is RF5 on Dev board) (13 is Rx and 15 is Tx on Wifi)

    U2MODEbits.ON = 1;
    U2MODEbits.BRGH = 0;
    U2STAbits.URXEN = 1;
    U2STAbits.UTXEN = 1;
    U2BRG=51; // Setting baud rate to 9600 manually (BRGH=0)

    // Set up UART3 = GPS connection

    U3MODEbits.ON = 1;
    U3MODEbits.BRGH = 0;
    U3STAbits.URXEN = 1;
    U3STAbits.UTXEN = 1;
    U3BRG=51; // Setting baud rate to 9600 manually (BRGH=0)

    // Setting up UART4 WIFI (Rx is D9, Tx is D1 on Dev board)

    U4MODEbits.ON = 1;
    U4MODEbits.BRGH = 0;
    U4STAbits.URXEN = 1;
    U4STAbits.UTXEN = 1;
    U4BRG=51; // Setting baud rate to 9600 manually (BRGH=0)

}

void timer_setup(void){

    __builtin_enable_interrupts();
    INTCONbits.MVEC = 1;

    T2CON = 0x0;
    T3CON = 0x0;
    T2CONSET = 0x0008;
    TMR2 = 0x0; // Clear contents of the TMR4 and TMR5
    PR2 = 0x0FFFFFFF; // Load PR4 and PR5 registers with 32-bit value
    IFS0bits.T3IF = 0; // Clear the Timer5 interrupt status flag
    IEC0bits.T3IE = 1;
    #define Time3 IFS0bits.T3IF

    T4CON = 0x0;
    T5CON = 0x0;
    T4CONSET = 0x0008;
    TMR4 = 0x0; // Clear contents of the TMR4 and TMR5
    PR4 = 0x0FFFFFFF; // Load PR4 and PR5 registers with 32-bit value
    IPC5bits.T5IP = 2;
    IPC5bits.IC5IS= 1;

    IFS0bits.T5IF = 0; // Clear the Timer5 interrupt status flag
    IEC0bits.T5IE = 1;

    #define GPStimer5 IFS0bits.T5IF

}

void define_function(void){

    // Defining our MOSFET switches (1 = OFF!!)
    TRISEbits.TRISE4=0; //set E4 to output
    TRISEbits.TRISE3=0; //set E3 to output
    TRISEbits.TRISE2=0; //set E2 to output

    #define wifiswitch LATEbits.LATE4
    #define GPSswitch LATEbits.LATE3
    #define RFIDswitch LATEbits.LATE2
    #define RXfullbufferRFID U2STAbits.URXDA
    #define RFIDoverrun U2STAbits.OERR
    #define GPSoverrun U3STAbits.OERR
    #define RXfullbufferGPS U3STAbits.URXDA
    #define GPSoverrun U3STAbits.OERR
    #define RXfullbufferGPS U3STAbits.URXDA
    #define RXfullbufferWiFi U4STAbits.URXDA
    #define WiFioverrun U4STAbits.OERR

}

// Global variables (for the interrupt subroutine)
unsigned char GPSdata[150];
int GPSfinish=0;
int q=0;
int r=0;
unsigned int save1=0;
unsigned int save2=1;
unsigned char save1array[100];
unsigned char save2array[100];

// Main Function
```

```
int main(int argc, char** argv) {

    wifiswitch=1; // Ensure everything off to start
    GPSswitch=1; // Off
    RFIDswitch=1; // Ensure everything off to start

    //Initialize UARTs/timers/define statements

    UART_init();
    timer_setup();
    define_function();

    // Defining Variables/Arrays
    unsigned char data1[150];
    unsigned char data2[150];
    int i=0;
    int j=0;

    unsigned char RFIDtag[20];
    int RFIDcount=0;
    int readerror=0;

    // This is our System test
    printf("\n\nThis is our System test\n");

    //Turning on GPS right now

    GPSswitch=0; // Turning on GPS

    printf("Waiting for Start Up Acks\n");

    // Wait for start up Acknowledgements
    GPSoverrun=0;
    i=0;
    j=0;
    while (1) {
        if (RXfullbufferGPS){
            data1[i] = U3RXREG;
            printf("%c", data1[i]);
            if(data1[i]=='\n'){
                if (j<4){j++;}
                else if (j==4) {break;}
            }
            i++;
        }
    }

    printf("Got GPS acks. GPS is started and searching.\n");

    printf("Command GPS to only give GGA data.\n");

    // Sending the GGA command to GPS
    char GGA[] = "$PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29\r\n";
    int loopGPS=51;
    i=0;
    while (i<loopGPS) {
        while (U3STAbits.UTXBF ==1) {}
        U3TXREG = GGA[i];
        i++;
    }

    //Wait for Acknowledgement of Command
    GPSoverrun=0;
    i=0;
    j=0;
    while (1) {
        if (RXfullbufferGPS){
            data1[i] = U3RXREG;
            printf("%c", data1[i]);
            if(data1[i]=='\n'){break;}
            i++;
        }
    }

    while (1) {
        if (RXfullbufferGPS){
            data1[i] = U3RXREG;
            printf("%c", data1[i]);
            if(data1[i]=='\n'){break;}
            i++;
        }
    }
    // Collect initial GPSdata/GPSfinish
        GPSoverrun=0;
        GPSfinish=0;
        j=0;
        while (1) {
            if (RXfullbufferGPS){
                GPSdata[GPSfinish] = U3RXREG;
                printf("%c", GPSdata[GPSfinish]);
                if(GPSdata[GPSfinish]=='\n'){
                    if (j==0){j++;GPSfinish=-1;}
                    else if (j==1) {break;}
                }
                GPSfinish++;
            }
        }

    // Main System Loop
    while(1){

        // Turn on GPS timer
        T4CONbits.ON = 1;

        // Turn on RFID
        RFIDswitch=0; //ON
        printf("\nReady for a Scan\n");

        // Wait for RFID scan
        RFIDoverrun=0;// Wipes out any superfluous data
        RFIDcount=0;
        while(1){
            if(RFIDoverrun){printf("\nOverrun\n");RFIDoverrun=0;T4CONbits.ON = 1;}
            if(RXfullbufferRFID){
                RFIDtag[RFIDcount] = U2RXREG;
                if(RFIDtag[RFIDcount]=='\n'){break;}
                RFIDcount++;
            }
        }

        printf("RFID scanned. RFIDCount = %d.\n\n",RFIDcount);
        T4CONbits.ON = 0;TMR4=0; GPStimer5=0; // Turn GPSTimer off while we process data
        // Turn off RFID
        RFIDswitch=1; //OFF

        // Writing to an Array to Send to WiFi
        int k=0;
        unsigned char exportWiFi[RFIDcount+GPSfinish+4];
        exportWiFi[0]='|';
```

```
        for (k=2;k<(RFIDcount);k++){exportWiFi[k]=RFIDtag[k-1];}
        exportWiFi[RFIDcount+1]=',';
        for
(k=(RFIDcount+2);k<(RFIDcount+1+GPSfinish+1+1);k++){exportWiFi[k]=GPSdata[k-(RFIDcount
+2)];}
        exportWiFi[RFIDcount+GPSfinish+1+1+1]='+';

        // Print to Terminal as a test
        for (k=3;k<(RFIDcount+GPSfinish+4);k++){
           if (k==3){printf("exportWiFi given below.\n\n");}
           printf("%c",exportWiFi[k]);
        }

        // Turn On WiFi
        wifiswitch=0; //ON

        // Wait for WiFi to acknowledge that it is connected
        unsigned char WiFiack[100];
        int WiFicount=-1;
        WiFioverrun = 0;
        int readerror=0;
        TMR2=0; Time3=0; // Resets Timer
        while(1){
           if(WiFicount==-1){WiFicount=0;T2CONbits.ON=1;printf("\nTimer has Started. Waiting for
WiFi Connection.\n");}
           if(RXfullbufferWiFi){
              if (WiFioverrun) {WiFioverrun = 0;}
              WiFiack[WiFicount] = U4RXREG;
              if(WiFiack[WiFicount]=='g'){printf("WiFi Connected!\n");break;}
              if(WiFiack[WiFicount]=='s'){printf("WiFi Not Connected!\n");break;}
              WiFicount++;
           }
           if (Time3){Time3=0;readerror=1;break;}
        }
        T2CONbits.ON=0;
        if(readerror==0){TMR2=0; Time3=0;}
        // In Case the WiFi timed out
        if(readerror==1){printf("\nError! WiFi Timeout.\n");wifiswitch=1;}
        //if(readerror==1){readerror=0;continue;} //From previous code, before the saving code


        // If WiFi didn't connect/timed out, saving the data and resetting
```

```
if(WiFiack[WiFicount]=='s'||readerror==1){
   printf("Now saving read data.\n");
   // Saving to next open array
   if((save1==1)&&(save2==1)){save1=0;}


   if (save2==0){
      for (k=0;k<(RFIDcount+GPSfinish+4);k++){
         {
            save2array[k]=exportWiFi[k];
            // Print to terminal as test
            if (k==0){printf("Saved version of exportWiFi given below.\n");}
            printf("%c",save2array[k]);
         }
      }
      save2=1;
   }

   if (save1==0){
      for (k=0;k<(RFIDcount+GPSfinish+4);k++){
         {
            save1array[k]=exportWiFi[k];
            // Print to terminal as test
            if (k==0){printf("Saved version of exportWiFi given below.\n");}
            printf("%c",save1array[k]);
         }
      }
      save1=1;save2=0;
   }

   if (readerror==1){readerror=0;}
   printf("Data now saved in array. Reseting to the Top.\n");
   continue;
}

printf("\nSending Scanned Array to Wifi.\n");

// Sending that array to the WiFi Chip
i=3;
while (i<(RFIDcount+GPSfinish+4)) {
   while (U4STAbits.UTXBF ==1) {}
   U4TXREG = exportWiFi[i];
```

```
            printf("%c",exportWiFi[i]);
            i++;
            if(i==15){i=16;}
        }

        printf("Done Sending to Wifi.\n");

        // Wait for WiFi to send back

        unsigned char WiFitest[100];
        WiFicount=-1;
        WiFioverrun = 0;
        readerror=0;
        TMR2=0; Time3=0; // Resets Timer Stuff
        while(1){
            if(WiFicount==-1){WiFicount=0;T2CONbits.ON=1;printf("\nTimer has Started. Waiting for
WiFi Acknowledgement of the Send.\n");}
            if(RXfullbufferWiFi){
                if (Time3){Time3=0;readerror=1;break;}
                if (WiFioverrun) {WiFioverrun = 0;}
                WiFitest[WiFicount] = U4RXREG;
                if(WiFitest[WiFicount]=='z'){break;}
                WiFicount++;
            }
            if (Time3){Time3=0;readerror=1;break;}
        }
        T2CONbits.ON=0;
        if(readerror==1){printf("\nError! WiFi Timeout, never acknowledged the
send.\n");wifiswitch=1;readerror=0;continue;}
        if(readerror==0){TMR2=0; Time3=0;}

        printf("\nWiFi acknowledged. Reset to the Top.\n");

        // Turn Off WiFi
        wifiswitch=1; //OFF

    }

    return (EXIT_SUCCESS);
}

void __ISR(20, IPL2AUTO)
    GPS2update(void)
```

```
{
    printf("In the time 5 interrupt. Collecting GGA data.\n");
    delay_ms(100);

    unsigned char data1[150];
    unsigned char data2[150];
    int i=0;
    int j=0;


    // Collect one line of GPS data back
        GPSoverrun=0;
        int GPScount=0;
        j=0;
        while (1) {
            if (RXfullbufferGPS){
                data2[GPScount] = U3RXREG;
                printf("%c", data2[GPScount]);
                if(data2[GPScount]=='\n'){
                    if (j==0){j++;GPScount=-1;}
                    else if (j==1) {break;}
                }
                GPScount++;
            }
        }

        printf("GPS received. GPSfinish = %d.\n\n",GPScount);

        if(GPScount>45)
        { for (j=0;j<GPScount;j++){GPSdata[j]=data2[j];}
        GPSfinish=GPScount;
        }

        // Checking WiFi to see if we can send saved data

    int qmax=2;

    if (save1&&q>qmax)
        {
        // Check for WiFi
        wifiswitch=0; // Turn on WiFi

        unsigned char WiFiack[100];
        int WiFicount=-1;
        WiFioverrun = 0;
        int readerror=0;
        TMR2=0; Time3=0; // Resets Timer
        while(1){
           if(WiFicount==-1){WiFicount=0;T2CONbits.ON=1;printf("\nTimer has Started. Waiting for
WiFi Connection.\n");}
           if(RXfullbufferWiFi){
              if (WiFioverrun) {WiFioverrun = 0;}
              WiFiack[WiFicount] = U4RXREG;
              if(WiFiack[WiFicount]=='g'){printf("WiFi Connected!\n");break;}
              if(WiFiack[WiFicount]=='s'){printf("WiFi Not Connected!\n");break;}
              WiFicount++;
           }
           if (Time3){Time3=0;readerror=1;break;}
        }
        T2CONbits.ON=0;
        if(readerror==0){TMR2=0; Time3=0;}
        // In Case the WiFi timed out
        if(readerror==1){printf("\nError! WiFi Timeout.\n");wifiswitch=1;}


        // If Wifi is found, send it

           if(WiFiack[WiFicount]=='g')
           {

              printf("\nSending Scanned Array to Wifi.\n");

                 // Sending that array to the WiFi Chip

              if (save1){
                 i=3;
                 while (i<(15+GPSfinish+4)) {
                 while (U4STAbits.UTXBF ==1) {}
                 U4TXREG = save1array[i];
                 printf("%c",save1array[i]);
                 i++;
                 if(i==15){i=16;}
               }
                 save1=0;

                 // Wait for WiFi to send back
```

```
                    unsigned char WiFitest[100];
                    WiFicount=-1;
                    WiFioverrun = 0;
                    readerror=0;
                    TMR2=0; Time3=0; // Resets Timer Stuff
                    while(1){
                        if(WiFicount==-1){WiFicount=0;T2CONbits.ON=1;printf("\nTimer has Started.
Waiting for WiFi Acknowledgement on sending array1.\n");}
                        if(RXfullbufferWiFi){
                            if (Time3){Time3=0;readerror=1;break;}
                            if (WiFioverrun) {WiFioverrun = 0;}
                            WiFitest[WiFicount] = U4RXREG;
                            if(WiFitest[WiFicount]=='z'){break;}
                            WiFicount++;
                        }
                        if (Time3){Time3=0;readerror=1;break;}
                    }
                    T2CONbits.ON=0;
                    if(readerror==1){printf("\nError! WiFi Timeout, never acknowledged the
send.\n");wifiswitch=1;}
                    if(readerror==0){TMR2=0; Time3=0;save1=0;q=0;printf("q is now reset to zero.\n");}
//Reset looping variable

                    printf("\nWiFi acknowledged on sending array1\n");


                }

                if (save2){
                    i=3;
                    while (i<(15+GPSfinish+4)) {
                    while (U4STAbits.UTXBF ==1) {}
                    U4TXREG = save2array[i];
                    printf("%c",save2array[i]);
                    i++;
                    if(i==15){i=16;}
                 }
                    if(!save1){save2=1;}
                    else {save2=0;}


                  printf("Done Sending to Wifi.\n");
                // Wait for WiFi to send back

                  unsigned char WiFitest[100];
                  WiFicount=-1;
                  WiFioverrun = 0;
                  readerror=0;
                  TMR2=0; Time3=0; // Resets Timer Stuff
                  while(1){
                     if(WiFicount==-1){WiFicount=0;T2CONbits.ON=1;printf("\nTimer has Started.
Waiting for WiFi Acknowledgement of the Send.\n");}
                     if(RXfullbufferWiFi){
                        if (Time3){Time3=0;readerror=1;break;}
                        if (WiFioverrun) {WiFioverrun = 0;}
                        WiFitest[WiFicount] = U4RXREG;
                        if(WiFitest[WiFicount]=='z'){break;}
                        WiFicount++;
                     }
                     if (Time3){Time3=0;readerror=1;break;}
                  }
                  T2CONbits.ON=0;
                  if(readerror==1){printf("\nError! WiFi Timeout, never acknowledged the
send.\n");wifiswitch=1;}
                  if(readerror==0){TMR2=0; Time3=0;save1=0;q=0;printf("q is now reset to zero.\n");}
//Reset looping variable

                  printf("\nWiFi acknowledged on sending array2.\n");
                }
            }
        }

    else {if(q>qmax){q=0;}q++;printf("q = %i. Leaving interrupt function.\n",q);}

    wifiswitch=1; // Wifi off


    // Checking to see if sending backpack

    int rmax=1; // Define how long to wait for backpack to automatically update
    char backpack[] = "  TRAKPAK SPOT ";
    int b=15;

    if(r>rmax){
```

```
 // Writing Backpack Array to Send to WiFi
int k=0;
unsigned char exportWiFi[b+GPSfinish+4];
for (k=2;k<(b);k++){exportWiFi[k]=backpack[k-1];}
exportWiFi[b+1]=',';
for (k=(b+2);k<(b+1+GPSfinish+1+1);k++){exportWiFi[k]=GPSdata[k-(b+2)];}
exportWiFi[b+GPSfinish+1+1+1]='+';

// Print to Terminal as a test
for (k=3;k<(b+GPSfinish+4);k++){
   if (k==3){printf("Backpack array given below.\n\n");}
   printf("%c",exportWiFi[k]);
}

// Turn On WiFi
wifiswitch=0; //ON

// Wait for WiFi to acknowledge that it is connected
unsigned char WiFiack[100];
int WiFicount=-1;
WiFioverrun = 0;
int readerror=0;
TMR2=0; Time3=0; // Resets Timer
while(1){
   if(WiFicount==-1){WiFicount=0;T2CONbits.ON=1;printf("\nTimer has Started. Waiting for
WiFi Connection.\n");}
   if(RXfullbufferWiFi){
      if (WiFioverrun) {WiFioverrun = 0;}
      WiFiack[WiFicount] = U4RXREG;
      if(WiFiack[WiFicount]=='g'){printf("WiFi Connected!\n");break;}
      if(WiFiack[WiFicount]=='s'){printf("WiFi Not Connected!\n");break;}
      WiFicount++;
   }
   if (Time3){Time3=0;readerror=1;break;}
}
T2CONbits.ON=0;
if(readerror==0){TMR2=0; Time3=0;}
// In Case the WiFi timed out
if(readerror==1){printf("\nError! WiFi Timeout.\n");wifiswitch=1;}
//if(readerror==1){readerror=0;continue;} //From previous code, before the saving code
```

```
// If WiFi didn't connect/timed out, saving the data and resetting


if(WiFiack[WiFicount]=='s'||readerror==1){
    printf("Now saving read data.\n");
    // Saving to next open array
    if((save1==1)&&(save2==1)){save1=0;}


    if (save2==0){
        for (k=0;k<(b+GPSfinish+4);k++){
            {
                save2array[k]=exportWiFi[k];
                // Print to terminal as test
                if (k==0){printf("Saved version of exportWiFi given below.\n");}
                printf("%c",save2array[k]);
            }
        }
        save2=1;
    }

    if (save1==0){
        for (k=0;k<(b+GPSfinish+4);k++){
            {
                save1array[k]=exportWiFi[k];
                // Print to terminal as test
                if (k==0){printf("Saved version of exportWiFi given below.\n");}
                printf("%c",save1array[k]);
            }
        }
        save1=1;save2=0;
    }

    if (readerror==1){readerror=0;}
    printf("Data now saved in array. Reseting to the Top.\n");
}
else {
    printf("\nSending Backpack Array to Wifi.\n");

    // Sending that array to the WiFi Chip
    i=3;
    while (i<(b+GPSfinish+4)) {
        while (U4STAbits.UTXBF ==1) {}
```

```
            U4TXREG = exportWiFi[i];
            printf("%c",exportWiFi[i]);
            i++;
            if(i==15){i=16;}
        }

        printf("\nDone Sending Backpack to Wifi.\n");

        // Wait for WiFi to send back

        unsigned char WiFitest[100];
        WiFicount=-1;
        WiFioverrun = 0;
        readerror=0;
        TMR2=0; Time3=0; // Resets Timer Stuff
        while(1){
            if(WiFicount==-1){WiFicount=0;T2CONbits.ON=1;printf("\nTimer has Started. Waiting
for WiFi Acknowledgement of sending Backpack GPS.\n");}
            if(RXfullbufferWiFi){
                if (Time3){Time3=0;readerror=1;break;}
                if (WiFioverrun) {WiFioverrun = 0;}
                WiFitest[WiFicount] = U4RXREG;
                if(WiFitest[WiFicount]=='z'){break;}
                WiFicount++;
            }
            if (Time3){Time3=0;readerror=1;break;}
        }
        T2CONbits.ON=0;
        if(readerror==1){printf("\nError! WiFi Timeout, never acknowledged the
send.\n");wifiswitch=1;readerror=0;}
        else if(readerror==0){TMR2=0; Time3=0;r=0;}

        printf("\nWiFi acknowledged. Leaving Interrupt Function\n");

        // Turn Off WiFi
        wifiswitch=1; //OFF
      }
    }
    else {if(r>rmax){r=0;}r++;printf("r = %i. Leaving interrupt function.\n",r);}


    GPStimer5= 0; //Set flag low
```

```
}
```

## 8.2.2 Mobile App Code

**ANDROID MANIFEST:**

**Android Manifest**

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="org.eclipse.paho.android.service.sample"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="19"
      android:targetSdkVersion="21" />

   <!-- Permissions the Application Requires -->
   <uses-permission android:name="android.permission.WAKE_LOCK" />
   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >

      <!-- Mqtt Service -->
      <service android:name="org.eclipse.paho.android.service.MqttService" >
      </service>

      <!--  Connection Details Activity -->
      <activity
         android:name="org.eclipse.paho.android.service.sample.ConnectionDetails"
         >//android:label="@string/title_activity_connection_details">

      </activity>

      <!-- Main Activity -->
      <activity android:name="org.eclipse.paho.android.service.sample.ClientConnections" >
```

```
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<!-- New Connection Activity -->
<activity
    android:name="org.eclipse.paho.android.service.sample.NewConnection"
    android:label="@string/title_activity_new_connection" >
</activity>


<!-- Advanced Options Screen -->
<activity
    android:name="org.eclipse.paho.android.service.sample.Advanced"
    android:label="@string/title_activity_advanced" >
</activity>

<!-- Last Will Activity -->
<activity
    android:name="org.eclipse.paho.android.service.sample.LastWill"
    android:label="@string/title_activity_last_will" >
</activity>

<activity
    android:name="org.eclipse.paho.android.service.sample.notification"
    android:label="@string/title_activity_notification"
    >
</activity>
<activity android:name="org.eclipse.paho.android.service.sample.MapsActivity"
    android:label="@string/title_maps_activity"
    >
</activity>
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version"
    />
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value= "AIzaSyB5eSIRErRjNQhEKzGMsGvVX6QzSEGoF5U"
    />
    </application>

</manifest>
```

**JAVA FILES:**

**Action Listener**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.content.Context;
import android.util.Log;
import android.widget.Toast;

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.android.service.sample.Connection.ConnectionStatus;
import org.eclipse.paho.client.mqttv3.IMqttActionListener;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttSecurityException;

/**
 * This Class handles receiving information from the
 * {@link MqttAndroidClient} and updating the {@link Connection} associated with
 * the action
 */
class ActionListener implements IMqttActionListener {

  /**
```

```
             * Actions that can be performed Asynchronously <strong>and</strong> associated
with a
             * {@link ActionListener} object
             *
             */
            enum Action {
              /** Connect Action **/
              CONNECT,
              /** Disconnect Action **/
              DISCONNECT,
              /** Subscribe Action **/
              SUBSCRIBE,
              /** Publish Action **/
              PUBLISH
            }

            /**
             * The {@link Action} that is associated with this instance of
             * <code>ActionListener</code>
             **/
            private Action action;
            /** The arguments passed to be used for formatting strings**/
            private String[] additionalArgs;
            /** Handle of the {@link Connection} this action was being executed on **/
            private String clientHandle;
            /** {@link Context} for performing various operations **/
            private Context context;

            /**
             * Creates a generic action listener for actions performed form any activity
             *
             * @param context
             *          The application context
             * @param action
             *          The action that is being performed
             * @param clientHandle
             *          The handle for the client which the action is being performed
             *          on
             * @param additionalArgs
             *          Used for as arguments for string formating
             */
            public ActionListener(Context context, Action action,
               String clientHandle, String... additionalArgs) {
```

```
        this.context = context;
        this.action = action;
        this.clientHandle = clientHandle;
        this.additionalArgs = additionalArgs;
      }

      /**
       * The action associated with this listener has been successful.
       *
       * @param asyncActionToken
       *            This argument is not used
       */
      @Override
      public void onSuccess(IMqttToken asyncActionToken) {
        switch (action) {
          case CONNECT :
            connect();
            String topic = "trakPak";
            int qos = 1;
            try{

Connections.getInstance(context).getConnection(clientHandle).getClient().subscribe(topic, qos,
null, new ActionListener(context, Action.SUBSCRIBE, clientHandle, topic));
            }
            catch (MqttSecurityException e){
              Log.e(this.getClass().getCanonicalName(), "Failed to subscribe to" + topic + " the
client with the handle " + clientHandle, e);
            }
            catch (MqttException e) {
              Log.e(this.getClass().getCanonicalName(), "Failed to subscribe to" + topic + " the
client with the handle " + clientHandle, e);
            }
            break;
          case DISCONNECT :
            disconnect();
            break;
          case SUBSCRIBE :
            subscribe();
            break;
          case PUBLISH :
            publish();
            break;
        }
```

```
}

/**
 * A publish action has been successfully completed, update connection
 * object associated with the client this action belongs to, then notify the
 * user of success
 */
private void publish() {

  Connection c = Connections.getInstance(context).getConnection(clientHandle);
  String actionTaken = context.getString(R.string.toast_pub_success,
      (Object[]) additionalArgs);
  c.addAction(actionTaken);
  Notify.toast(context, actionTaken, Toast.LENGTH_SHORT);
}

/**
 * A subscribe action has been successfully completed, update the connection
 * object associated with the client this action belongs to and then notify
 * the user of success
 */
private void subscribe() {
  Connection c = Connections.getInstance(context).getConnection(clientHandle);
  String actionTaken = context.getString(R.string.toast_sub_success,
      (Object[]) additionalArgs);
  c.addAction(actionTaken);
  Notify.toast(context, actionTaken, Toast.LENGTH_SHORT);

}

/**
 * A disconnection action has been successfully completed, update the
 * connection object associated with the client this action belongs to and
 * then notify the user of success.
 */
private void disconnect() {
  Connection c = Connections.getInstance(context).getConnection(clientHandle);
  c.changeConnectionStatus(ConnectionStatus.DISCONNECTED);
  String actionTaken = context.getString(R.string.toast_disconnected);
  c.addAction(actionTaken);

}
```

```
/**
 * A connection action has been successfully completed, update the
 * connection object associated with the client this action belongs to and
 * then notify the user of success.
 */
private void connect() {

  Connection c = Connections.getInstance(context).getConnection(clientHandle);
  c.changeConnectionStatus(Connection.ConnectionStatus.CONNECTED);
  c.addAction("Client Connected");

}

/**
 * The action associated with the object was a failure
 *
 * @param token
 *            This argument is not used
 * @param exception
 *            The exception which indicates why the action failed
 */
@Override
public void onFailure(IMqttToken token, Throwable exception) {
  switch (action) {
    case CONNECT :
      connect(exception);
      break;
    case DISCONNECT :
      disconnect(exception);
      break;
    case SUBSCRIBE :
      subscribe(exception);
      break;
    case PUBLISH :
      publish(exception);
      break;
  }

}

/**
 * A publish action was unsuccessful, notify user and update client history
 *
 * @param exception
 *           This argument is not used
 */
private void publish(Throwable exception) {
  Connection c = Connections.getInstance(context).getConnection(clientHandle);
  String action = context.getString(R.string.toast_pub_failed,
      (Object[]) additionalArgs);
  c.addAction(action);
  Notify.toast(context, action, Toast.LENGTH_SHORT);

}

/**
 * A subscribe action was unsuccessful, notify user and update client history
 * @param exception This argument is not used
 */
private void subscribe(Throwable exception) {
  Connection c = Connections.getInstance(context).getConnection(clientHandle);
  String action = context.getString(R.string.toast_sub_failed,
      (Object[]) additionalArgs);
  c.addAction(action);
  Notify.toast(context, action, Toast.LENGTH_SHORT);

}

/**
 * A disconnect action was unsuccessful, notify user and update client history
 * @param exception This argument is not used
 */
private void disconnect(Throwable exception) {
  Connection c = Connections.getInstance(context).getConnection(clientHandle);
  c.changeConnectionStatus(ConnectionStatus.DISCONNECTED);
  c.addAction("Disconnect Failed - an error occured");

}

/**
 * A connect action was unsuccessful, notify the user and update client history
 * @param exception This argument is not used
 */
private void connect(Throwable exception) {
  Connection c = Connections.getInstance(context).getConnection(clientHandle);
```

```
    c.changeConnectionStatus(Connection.ConnectionStatus.ERROR);
    c.addAction("Client failed to connect");

  }

}
```

**Activity Constants**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import org.eclipse.paho.client.mqttv3.MqttMessage;

/**
 * This Class provides constants used for returning results from an activity
 *
 */
public class ActivityConstants {

  /** Application TAG for logs where class name is not used*/
  static final String TAG = "MQTT Android";

  /*Default values **/

  /** Default QOS value*/
  static final int defaultQos = 1;
  /** Default timeout*/
  static final int defaultTimeOut = 1000;
  /** Default keep alive value*/
  static final int defaultKeepAlive = 10;
  /** Default SSL enabled flag*/
  static final boolean defaultSsl = false;
```

```
/** Default message retained flag */
static final boolean defaultRetained = false;
/** Default last will message*/
static final MqttMessage defaultLastWill = null;
/** Default port*/
static final int defaultPort = 1883;

/** Connect Request Code */
static final int connect = 0;
/** Advanced Connect Request Code  **/
static final int advancedConnect = 1;
/** Last will Request Code  **/
static final int lastWill = 2;
/** Show History Request Code  **/
static final int showHistory = 3;

/* Bundle Keys */

/** Server Bundle Key **/
static final String server = "server";
/** Port Bundle Key **/
static final String port = "port";
/** ClientID Bundle Key **/
static final String clientId = "clientId";
/** Topic Bundle Key **/
static final String topic = "topic";
/** History Bundle Key **/
static final String history = "history";
/** Message Bundle Key **/
static final String message = "message";
/** Retained Flag Bundle Key **/
static final String retained = "retained";
/** QOS Value Bundle Key **/
static final String qos = "qos";
/** User name Bundle Key **/
static final String username = "username";
/** Password Bundle Key **/
static final String password = "password";
/** Keep Alive value Bundle Key **/
static final String keepalive = "keepalive";
/** Timeout Bundle Key **/
static final String timeout = "timeout";
/** SSL Enabled Flag Bundle Key **/
```

```
    static final String ssl = "ssl";
    /** SSL Key File Bundle Key **/
    static final String ssl_key = "ssl_key";
    /** Connections Bundle Key **/
    static final String connections = "connections";
    /** Clean Session Flag Bundle Key **/
    static final String cleanSession = "cleanSession";
    /** Action Bundle Key **/
    static final String action = "action";

    /* Property names */

    /** Property name for the history field in {@link Connection} object for use with {@link
java.beans.PropertyChangeEvent} **/
    static final String historyProperty = "history";

    /** Property name for the connection status field in {@link Connection} object for use
with {@link java.beans.PropertyChangeEvent} **/
    static final String ConnectionStatusProperty = "connectionStatus";

    /* Useful constants*/

    /** Space String Literal **/
    static final String space = " ";
    /** Empty String for comparisons **/
    static final String empty = new String();


}
```

**Advanced**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
```

```
package org.eclipse.paho.android.service.sample;

import android.app.Activity;
import android.app.Dialog;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MenuItem.OnMenuItemClickListener;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;

import java.util.HashMap;
import java.util.Map;

/**
 * Advanced connection options activity
 *
 */
@SuppressWarnings("ALL")
public class Advanced extends Activity {

  /**
   * Reference to this class used in {@link Advanced.Listener} methods
   */
  private Advanced advanced = this;
  /**
   * Holds the result data from activities launched from this activity
   */
  private Bundle resultData = null;

  private int openfileDialogId = 0;

  /**
   * @see Activity#onCreate(Bundle)
   */
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
          setContentView(R.layout.activity_advanced);

          ((Button) findViewById(R.id.sslKeyBut)).setOnClickListener(new OnClickListener(){

                    @Override
                    public void onClick(View v) {
                              //showFileChooser();
                              showDialog(openfileDialogId);
                    }});

          ((CheckBox) findViewById(R.id.sslCheckBox)).setOnClickListener(new
OnClickListener(){

                    @Override
                    public void onClick(View v) {
                              if(((CheckBox)v).isChecked())
                              {
                                        ((Button)findViewById(R.id.sslKeyBut)).setClickable(true);
                              }else
                              {
                                        ((Button)findViewById(R.id.sslKeyBut)).setClickable(false);
                              }

                    }});

          ((Button)findViewById(R.id.sslKeyBut)).setClickable(false);
        }

        /**
         * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
         */
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
          getMenuInflater().inflate(R.menu.activity_advanced, menu);

          Listener listener = new Listener();
          menu.findItem(R.id.setLastWill).setOnMenuItemClickListener(listener);
          menu.findItem(R.id.ok).setOnMenuItemClickListener(listener);

          return true;
        }

        /**
```

```
 * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
 */
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case android.R.id.home :
      NavUtils.navigateUpFromSameTask(this);
      return true;
  }
  return super.onOptionsItemSelected(item);
}

/**
 * @see android.app.Activity#onActivityResult(int, int, android.content.Intent)
 */
@Override
protected void onActivityResult(int requestCode, int resultCode,
    Intent intent) {
  // get the last will data
  if (resultCode == RESULT_CANCELED) {
    return;
  }
  resultData = intent.getExtras();

}

        /**
         * @see android.app.Activity#onCreateDialog(int)
         */
        @Override
        protected Dialog onCreateDialog(int id) {
                if (id == openfileDialogId) {
                        Map<String, Integer> images = new HashMap<String, Integer>();
                        images.put(OpenFileDialog.sRoot, R.drawable.ic_launcher);
                        images.put(OpenFileDialog.sParent, R.drawable.ic_launcher);
                        images.put(OpenFileDialog.sFolder, R.drawable.ic_launcher);
                        images.put("bks", R.drawable.ic_launcher);
                        images.put(OpenFileDialog.sEmpty, R.drawable.ic_launcher);
                        Dialog dialog = OpenFileDialog.createDialog(id, this, "openfile",
                                new CallbackBundle() {
                                        @Override
                                        public void callback(Bundle bundle) {
```

```
                                                            String filepath =
bundle.getString("path");
                                                            // setTitle(filepath);
                                                            ((EditText)
findViewById(R.id.sslKeyLocaltion))
                                                                        .setText(filepath);
                                                      }
                                                }, ".bks;", images);
                                    return dialog;
                              }
                              return null;
                        }

                  /**
                   * Deals with button clicks for the advanced options page
                   *
                   */
                  private class Listener implements OnMenuItemClickListener {

                    /**
                     * @see
android.view.MenuItem.OnMenuItemClickListener#onMenuItemClick(MenuItem)
                     */
                    @Override
                    public boolean onMenuItemClick(MenuItem item) {

                      int button = item.getItemId();

                      switch (button) {
                        case R.id.ok :
                          ok();
                          break;

                        case R.id.setLastWill :
                          lastWill();
                          break;
                      }
                      return false;
                    }

                    /**
                     * Packs the default options into an intent
                     * @return intent packed with default options
```

```
 */
@SuppressWarnings("unused")
private Intent packDefaults() {
  Intent intent = new Intent();

  // check to see if there is any result data if there is not any
  // result data build some with defaults

  intent.putExtras(resultData);
  intent.putExtra(ActivityConstants.username, ActivityConstants.empty);
  intent.putExtra(ActivityConstants.password, ActivityConstants.empty);

  intent.putExtra(ActivityConstants.timeout, ActivityConstants.defaultTimeOut);
  intent.putExtra(ActivityConstants.keepalive,
      ActivityConstants.defaultKeepAlive);
  intent.putExtra(ActivityConstants.ssl, ActivityConstants.defaultSsl);

  return intent;
}

/**
 *  Starts an activity to collect last will options
 */
private void lastWill() {

  Intent intent = new Intent();
  intent.setClassName(advanced, "org.eclipse.paho.android.service.sample.LastWill");
  advanced.startActivityForResult(intent, ActivityConstants.lastWill);

}

/**
 * Packs all the options the user has chosen, along with defaults the user has not
chosen
 */
private void ok() {

  int keepalive;
  int timeout;

  Intent intent = new Intent();

  if (resultData == null) {
```

```
  resultData = new Bundle();
  resultData.putString(ActivityConstants.message, ActivityConstants.empty);
  resultData.putString(ActivityConstants.topic, ActivityConstants.empty);
  resultData.putInt(ActivityConstants.qos, ActivityConstants.defaultQos);
  resultData.putBoolean(ActivityConstants.retained,
      ActivityConstants.defaultRetained);
}

intent.putExtras(resultData);

// get all advance options
String username = ((EditText) findViewById(R.id.uname)).getText()
    .toString();
String password = ((EditText) findViewById(R.id.password))
    .getText().toString();
String sslkey = null;
boolean ssl = ((CheckBox) findViewById(R.id.sslCheckBox)).isChecked();
if(ssl)
{
       sslkey = ((EditText) findViewById(R.id.sslKeyLocaltion))
              .getText().toString();
}
try {
  timeout = Integer
      .parseInt(((EditText) findViewById(R.id.timeout))
         .getText().toString());
}
catch (NumberFormatException nfe) {
  timeout = ActivityConstants.defaultTimeOut;
}
try {
  keepalive = Integer
      .parseInt(((EditText) findViewById(R.id.keepalive))
         .getText().toString());
}
catch (NumberFormatException nfe) {
  keepalive = ActivityConstants.defaultKeepAlive;
}

//put the daya collected into the intent
intent.putExtra(ActivityConstants.username, username);
intent.putExtra(ActivityConstants.password, password);
```

```
      intent.putExtra(ActivityConstants.timeout, timeout);
      intent.putExtra(ActivityConstants.keepalive, keepalive);
      intent.putExtra(ActivityConstants.ssl, ssl);
      intent.putExtra(ActivityConstants.ssl_key, sslkey);
      //set the result as okay, with the data, and finish
      advanced.setResult(RESULT_OK, intent);
      advanced.finish();
    }

  }

}
```

**Callback Bundle**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;


import android.os.Bundle;

/**
 * For File selector to share data
 *
 */
public interface CallbackBundle {
	abstract void callback(Bundle bundle);
}
```

**Client Connections**

```
/*******************************************************************************
* Copyright (c) 1999, 2014 IBM Corp.
*
```

```
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.ActionMode;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.MenuItem.OnMenuItemClickListener;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemLongClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.android.service.sample.Connection.ConnectionStatus;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttSecurityException;

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Map;

/**
 * ClientConnections is the main activity for the sample application, it
 * displays all the active connections.
 *
 */
@SuppressWarnings("ALL")
public class ClientConnections extends ListActivity {

  /**
   * Token to pass to the MQTT Service
   */
  final static String TOKEN = "org.eclipse.paho.android.service.sample.ClientConnections";

  /**
   * ArrayAdapter to populate the list view
   */
  private ArrayAdapter<Connection> arrayAdapter = null;

  /**
   * {@link ChangeListener} for use with all {@link Connection} objects created by this instance
of <code>ClientConnections</code>
   */
  private ChangeListener changeListener = new ChangeListener();

  /**
   * This instance of <code>ClientConnections</code> used to update the UI in {@link
ChangeListener}
   */
  private ClientConnections clientConnections = this;

  /**
   * Contextual action bar active or not
   */
  private boolean contextualActionBarActive = false;

  /**
   * @see android.app.ListActivity#onCreate(Bundle)
   */
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ListView connectionList = getListView();
    connectionList.setOnItemLongClickListener(new LongClickItemListener());
    connectionList.setTextFilterEnabled(true);
    arrayAdapter = new ArrayAdapter<Connection>(this,
        R.layout.connection_text_view);
    setListAdapter(arrayAdapter);

    // get all the available connections
    Map<String, Connection> connections = Connections.getInstance(this)
        .getConnections();

    if (connections != null) {
      for (String s : connections.keySet())
      {
        arrayAdapter.add(connections.get(s));
      }
    }

  }

  /**
   * Creates the action bar for the activity
   *
   * @see ListActivity#onCreateOptionsMenu(Menu)
   */
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {

    OnMenuItemClickListener menuItemClickListener = new Listener(this);

    //load the correct menu depending on the status of logging
    if (Listener.logging)
    {
      getMenuInflater().inflate(R.menu.activity_connections_logging, menu);
      menu.findItem(R.id.endLogging).setOnMenuItemClickListener(menuItemClickListener);
    }
    else {
      getMenuInflater().inflate(R.menu.activity_connections, menu);
      menu.findItem(R.id.startLogging).setOnMenuItemClickListener(menuItemClickListener);
    }

    menu.findItem(R.id.newConnection).setOnMenuItemClickListener(
        menuItemClickListener);

    return true;
  }
```

```
/**
 * Listens for item clicks on the view
 *
 * @param listView
 *          The list view where the click originated from
 * @param view
 *          The view which was clicked
 * @param position
 *          The position in the list that was clicked
 */
@Override
protected void onListItemClick(ListView listView, View view, int position,
    long id) {
  super.onListItemClick(listView, view, position, id);

  if (!contextualActionBarActive) {
    Connection c = arrayAdapter.getItem(position);

    // start the connectionDetails activity to display the details about the
    // selected connection
    Intent intent = new Intent();
    intent.setClassName(getApplicationContext().getPackageName(),
        "org.eclipse.paho.android.service.sample.ConnectionDetails");
    intent.putExtra("handle", c.handle());
    startActivity(intent);
  }

}

/**
 * @see ListActivity#onActivityResult(int,int,Intent)
 */
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {

  if (resultCode == RESULT_CANCELED) {
    return;
  }

  Bundle dataBundle = data.getExtras();

  // perform connection create and connect
```

```
    connectAction(dataBundle);

  }

  /**
   * @see ListActivity#onResume()
   */
  @Override
  protected void onResume() {
    super.onResume();
    arrayAdapter.notifyDataSetChanged();

    //Recover connections.
    Map<String, Connection> connections = Connections.getInstance(this).getConnections();

    //Register receivers again
    for (Connection connection : connections.values()){
      connection.getClient().registerResources(this);
      connection.getClient().setCallback(new MqttCallbackHandler(this,
connection.getClient().getServerURI()+connection.getClient().getClientId()));
    }
  }

  /**
   * @see ListActivity#onDestroy()
   */
  @Override
  protected void onDestroy() {

    Map<String, Connection> connections = Connections.getInstance(this).getConnections();

    for (Connection connection : connections.values()){
            connection.registerChangeListener(changeListener);
            connection.getClient().unregisterResources();
    }
    super.onDestroy();
  }

  /**
   * Process data from the connect action
   *
   * @param data the {@link Bundle} returned by the {@link NewConnection} Acitivty
   */
  private void connectAction(Bundle data) {
    MqttConnectOptions conOpt = new MqttConnectOptions();
    /*
     * Mutal Auth connections could do something like this
     *
     *
     * SSLContext context = SSLContext.getDefault();
     * context.init({new CustomX509KeyManager()},null,null); //where CustomX509KeyManager
proxies calls to keychain api
     * SSLSocketFactory factory = context.getSSLSocketFactory();
     *
     * MqttConnectOptions options = new MqttConnectOptions();
     * options.setSocketFactory(factory);
     *
     * client.connect(options);
     *
     */

    // The basic client information
    String server = "senior-mqtt.esc.nd.edu";//(String) data.get(ActivityConstants.server);
    String clientId = (String) data.get(ActivityConstants.clientId);
    int port = 1883;//Integer.parseInt((String) data.get(ActivityConstants.port));
    boolean cleanSession = true;//(Boolean) data.get(ActivityConstants.cleanSession);

    boolean ssl = false;//(Boolean) data.get(ActivityConstants.ssl);
    String ssl_key = (String) data.get(ActivityConstants.ssl_key);
    String uri = null;
    if (ssl) {
      Log.e("SSLConnection", "Doing an SSL Connect");
      uri = "ssl://";

    }
    else {
      uri = "tcp://";
    }

    uri = uri + server + ":" + port;

    MqttAndroidClient client;
    client = Connections.getInstance(this).createClient(this, uri, clientId);

    if (ssl){
        try {
```

```
        if(ssl_key != null && !ssl_key.equalsIgnoreCase(""))
        {
                FileInputStream key = new FileInputStream(ssl_key);
                conOpt.setSocketFactory(client.getSSLSocketFactory(key,
                                            "mqtttest"));
        }

                } catch (MqttSecurityException e) {
                        Log.e(this.getClass().getCanonicalName(),
                        "MqttException Occured: ", e);
                } catch (FileNotFoundException e) {
                        Log.e(this.getClass().getCanonicalName(),
                        "MqttException Occured: SSL Key file not found", e);
                }
}

// create a client handle
String clientHandle = uri + clientId;

// last will message
String message = (String) data.get(ActivityConstants.message);
String topic = (String) data.get(ActivityConstants.topic);
Integer qos = (Integer) data.get(ActivityConstants.qos);
Boolean retained = (Boolean) data.get(ActivityConstants.retained);

// connection options

String username = (String) data.get(ActivityConstants.username);

String password = (String) data.get(ActivityConstants.password);

int timeout = (Integer) data.get(ActivityConstants.timeout);
int keepalive = (Integer) data.get(ActivityConstants.keepalive);

Connection connection = new Connection(clientHandle, clientId, server, port,
   this, client, ssl);
arrayAdapter.add(connection);

connection.registerChangeListener(changeListener);
// connect client

String[] actionArgs = new String[1];
actionArgs[0] = clientId;
```

```
connection.changeConnectionStatus(ConnectionStatus.CONNECTING);

conOpt.setCleanSession(cleanSession);
conOpt.setConnectionTimeout(timeout);
conOpt.setKeepAliveInterval(keepalive);
if (!username.equals(ActivityConstants.empty)) {
  conOpt.setUserName(username);
}
if (!password.equals(ActivityConstants.empty)) {
  conOpt.setPassword(password.toCharArray());
}

final ActionListener callback = new ActionListener(this,
    ActionListener.Action.CONNECT, clientHandle, actionArgs);

boolean doConnect = true;

if ((!message.equals(ActivityConstants.empty))
    || (!topic.equals(ActivityConstants.empty))) {
  // need to make a message since last will is set
  try {
    conOpt.setWill(topic, message.getBytes(), qos.intValue(),
        retained.booleanValue());
  }
  catch (Exception e) {
        Log.e(this.getClass().getCanonicalName(), "Exception Occured", e);
    doConnect = false;
    callback.onFailure(null, e);
  }
}
client.setCallback(new MqttCallbackHandler(this, clientHandle));


//set traceCallback
client.setTraceCallback(new MqttTraceCallback());

connection.addConnectionOptions(conOpt);
Connections.getInstance(this).addConnection(connection);
if (doConnect) {
  try {
    client.connect(conOpt, null, callback);
  }
  catch (MqttException e) {
        Log.e(this.getClass().getCanonicalName(),
            "MqttException Occured", e);
      }
    }

  }

  /**
   * <code>LongClickItemListener</code> deals with enabling and disabling the contextual
action bar and
   * processing the actions selected.
   *
   */
  private class LongClickItemListener implements OnItemLongClickListener,
ActionMode.Callback, OnClickListener {

    /** The index of the item selected, or -1 if an item is not selected **/
    private int selected = -1;
    /** The view of the item selected **/
    private View selectedView = null;
    /** The connection the view is representing **/
    private Connection connection = null;

    /* (non-Javadoc)
     * @see
android.widget.AdapterView.OnItemLongClickListener#onItemLongClick(android.widget.Adapte
rView, android.view.View, int, long)
     */
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id) {
      clientConnections.startActionMode(this);
      selected = position;
      selectedView = view;
      clientConnections.getListView().setSelection(position);
      view.setBackgroundColor(getResources().getColor(android.R.color.holo_blue_dark));
      return true;
    }

    /* (non-Javadoc)
     * @see android.view.ActionMode.Callback#onActionItemClicked(android.view.ActionMode,
android.view.MenuItem)
     */
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
      selectedView.setBackgroundColor(getResources().getColor(android.R.color.white));
      switch (item.getItemId()) {
        case R.id.delete :
          delete();
          mode.finish();
          return true;
        default :
          return false;
      }
    }

    /* (non-Javadoc)
     * @see android.view.ActionMode.Callback#onCreateActionMode(android.view.ActionMode,
android.view.Menu)
     */
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
      MenuInflater inflater = mode.getMenuInflater();
      inflater.inflate(R.menu.activity_client_connections_contextual, menu);
      clientConnections.contextualActionBarActive = true;
      return true;
    }

    /* (non-Javadoc)
     * @see android.view.ActionMode.Callback#onDestroyActionMode(android.view.ActionMode)
     */
    @Override
    public void onDestroyActionMode(ActionMode mode) {
      selected = -1;
      selectedView = null;

    }

    /* (non-Javadoc)
     * @see android.view.ActionMode.Callback#onPrepareActionMode(android.view.ActionMode,
android.view.Menu)
     */
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
      return false;
    }
```

```
  /**
   * Deletes the connection, disconnecting if required.
   */
  private void delete()
  {
    connection = arrayAdapter.getItem(selected);
    if (connection.isConnectedOrConnecting()) {

      //display a dialog
      AlertDialog.Builder builder = new AlertDialog.Builder(clientConnections);
      builder.setTitle(R.string.disconnectClient)
          .setMessage(getString(R.string.deleteDialog))
          .setNegativeButton(R.string.cancelBtn, new OnClickListener() {

            @Override
            public void onClick(DialogInterface arg0, int arg1) {
              //do nothing user cancelled action
            }
          })
          .setPositiveButton(R.string.continueBtn, this)
          .show();
    }
    else {
      arrayAdapter.remove(connection);
      Connections.getInstance(clientConnections).removeConnection(connection);
    }

  }

  @Override
  public void onClick(DialogInterface dialog, int which) {
    //user pressed continue disconnect client and delete
    try {
      connection.getClient().disconnect();
    }
    catch (MqttException e) {
      e.printStackTrace();
    }
    arrayAdapter.remove(connection);
    Connections.getInstance(clientConnections).removeConnection(connection);

  }
}
```

```
 /**
  * This class ensures that the user interface is updated as the Connection objects change their
states
  *
  *
  */
 private class ChangeListener implements PropertyChangeListener {

   /**
    * @see
java.beans.PropertyChangeListener#propertyChange(java.beans.PropertyChangeEvent)
    */
   @Override
   public void propertyChange(PropertyChangeEvent event) {

     if (!event.getPropertyName().equals(ActivityConstants.ConnectionStatusProperty)) {
       return;
     }
     clientConnections.runOnUiThread(new Runnable() {

       @Override
       public void run() {
         clientConnections.arrayAdapter.notifyDataSetChanged();
       }

     });

   }

 }
}
```

**Connection**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.content.Context;
import android.text.Html;
import android.text.Spanned;

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

/**
 *
 * Represents a {@link MqttAndroidClient} and the actions it has performed
 *
 */
public class Connection {

  /*
   * Basic Information about the client
   */
  /** ClientHandle for this Connection Object**/
  private String clientHandle = null;
  /** The clientId of the client associated with this <code>Connection</code> object **/
  private String clientId = null;
  /** The host that the {@link MqttAndroidClient} represented by this <code>Connection</code>
is represented by **/
  private String host = null;
  /** The port on the server this client is connecting to **/
  private int port = 0;
  /** {@link ConnectionStatus} of the {@link MqttAndroidClient} represented by this
<code>Connection</code> object. Default value is {@link ConnectionStatus#NONE} **/
  private ConnectionStatus status = ConnectionStatus.NONE;
  /** The history of the {@link MqttAndroidClient} represented by this <code>Connection</code>
object **/
  private ArrayList<String> history = null;
  /** The {@link MqttAndroidClient} instance this class represents**/
  private MqttAndroidClient client = null;

  /** Collection of {@link PropertyChangeListener} **/
  private ArrayList<PropertyChangeListener> listeners = new
ArrayList<PropertyChangeListener>();

  /** The {@link Context} of the application this object is part of**/
  private Context context = null;

  /** The {@link MqttConnectOptions} that were used to connect this client**/
  private MqttConnectOptions conOpt;

  /** True if this connection is secured using SSL **/
  private boolean sslConnection = false;

  /** Persistence id, used by {@link Persistence} **/
  private long persistenceId = -1;

  /**
   * Connections status for  a connection
   */
  enum ConnectionStatus {

    /** Client is Connecting **/
    CONNECTING,
    /** Client is Connected **/
    CONNECTED,
    /** Client is Disconnecting **/
    DISCONNECTING,
    /** Client is Disconnected **/
    DISCONNECTED,
    /** Client has encountered an Error **/
    ERROR,
    /** Status is unknown **/
    NONE
  }

  /**
   * Creates a connection from persisted information in the database store, attempting
   * to create a {@link MqttAndroidClient} and the client handle.
   * @param clientId The id of the client
   * @param host the server which the client is connecting to
   * @param port the port on the server which the client will attempt to connect to
   * @param context the application context
   * @param sslConnection true if the connection is secured by SSL
   * @return a new instance of <code>Connection</code>
   */
  public static Connection createConnection(String clientId, String host,
      int port, Context context, boolean sslConnection) {
    String handle = null;
    String uri = null;
    if (sslConnection) {
      uri = "ssl://" + host + ":" + port;
      handle = uri + clientId;
    }
    else {
      uri = "tcp://" + host + ":" + port;
      handle = uri + clientId;
    }
    MqttAndroidClient client = new MqttAndroidClient(context, uri, clientId);
    return new Connection(handle, clientId, host, port, context, client, sslConnection);

  }

  /**
   * Creates a connection object with the server information and the client
   * hand which is the reference used to pass the client around activities
   * @param clientHandle The handle to this <code>Connection</code> object
   * @param clientId The Id of the client
   * @param host The server which the client is connecting to
   * @param port The port on the server which the client will attempt to connect to
   * @param context The application context
   * @param client The MqttAndroidClient which communicates with the service for this
connection
   * @param sslConnection true if the connection is secured by SSL
   */
  public Connection(String clientHandle, String clientId, String host,
      int port, Context context, MqttAndroidClient client, boolean sslConnection) {
    //generate the client handle from its hash code
    this.clientHandle = clientHandle;
    this.clientId = clientId;
    this.host = host;
    this.port = port;
    this.context = context;
    this.client = client;
    this.sslConnection = sslConnection;
    history = new ArrayList<String>();
    StringBuffer sb = new StringBuffer();
    sb.append("Client: ");
    sb.append(clientId);
    sb.append(" created");
    addAction(sb.toString());
  }

  /**
   * Add an action to the history of the client
   * @param action the history item to add
   */
  public void addAction(String action) {

    Object[] args = new String[1];
    SimpleDateFormat sdf = new SimpleDateFormat(context.getString(R.string.dateFormat));
    args[0] = sdf.format(new Date());

    String timestamp = context.getString(R.string.timestamp, args);
    history.add(action + timestamp);

    notifyListeners(new PropertyChangeEvent(this, ActivityConstants.historyProperty, null, null));
  }

  /**
   * Generate an array of Spanned items representing the history of this
   * connection.
   *
   * @return an array of history entries
   */
  public Spanned[] history() {

    int i = 0;
    Spanned[] array = new Spanned[history.size()];

    for (String s : history) {
      if (s != null) {
        array[i] = Html.fromHtml(s);
        i++;
      }
    }
    return array;

  }

  /**
   * Gets the client handle for this connection
   * @return client Handle for this connection
   */
  public String handle() {
    return clientHandle;
  }

  /**
   * Determines if the client is connected
   * @return is the client connected
   */
  public boolean isConnected() {
    return status == ConnectionStatus.CONNECTED;
  }

  /**
   * Changes the connection status of the client
   * @param connectionStatus The connection status of this connection
   */
  public void changeConnectionStatus(ConnectionStatus connectionStatus) {
    status = connectionStatus;
    notifyListeners((new PropertyChangeEvent(this, ActivityConstants.ConnectionStatusProperty,
null, null)));
  }

  /**
   * A string representing the state of the client this connection
   * object represents
   *
   *
   * @return A string representing the state of the client
   */
  @Override
  public String toString() {
    StringBuffer sb = new StringBuffer();
    sb.append(clientId);
    sb.append("\n ");
  switch (status) {

    case CONNECTED :
      sb.append(context.getString(R.string.connectedto));
      break;
    case DISCONNECTED :
      sb.append(context.getString(R.string.disconnected));
      break;
    case NONE :
      sb.append(context.getString(R.string.no_status));
      break;
    case CONNECTING :
      sb.append(context.getString(R.string.connecting));
      break;
    case DISCONNECTING :
      sb.append(context.getString(R.string.disconnecting));
      break;
    case ERROR :
      sb.append(context.getString(R.string.connectionError));
  }
  sb.append(" ");
  sb.append(host);

  return sb.toString();
}

/**
 * Determines if a given handle refers to this client
 * @param handle The handle to compare with this clients handle
 * @return true if the handles match
 */
public boolean isHandle(String handle) {
  return clientHandle.equals(handle);
}

/**
 * Compares two connection objects for equality
 * this only takes account of the client handle
 * @param o The object to compare to
 * @return true if the client handles match
 */
@Override
public boolean equals(Object o) {
  }

    if (!(o instanceof Connection)) {
      return false;
    }

    Connection c = (Connection) o;

    return clientHandle.equals(c.clientHandle);

  }

  /**
   * Get the client Id for the client this object represents
   * @return the client id for the client this object represents
   */
  public String getId() {
    return clientId;
  }

  /**
   * Get the host name of the server that this connection object is associated with
   * @return the host name of the server this connection object is associated with
   */
  public String getHostName() {

    return host;
  }

  /**
   * Determines if the client is in a state of connecting or connected.
   * @return if the client is connecting or connected
   */
  public boolean isConnectedOrConnecting() {
    return (status == ConnectionStatus.CONNECTED) || (status ==
ConnectionStatus.CONNECTING);
  }

  /**
   * Client is currently not in an error state
   * @return true if the client is in not an error state
   */
  public boolean noError() {
    return status != ConnectionStatus.ERROR;
  }
```

```
/**
 * Gets the client which communicates with the android service.
 * @return the client which communicates with the android service
 */
public MqttAndroidClient getClient() {
  return client;
}

/**
 * Add the connectOptions used to connect the client to the server
 * @param connectOptions the connectOptions used to connect to the server
 */
public void addConnectionOptions(MqttConnectOptions connectOptions) {
  conOpt = connectOptions;

}

/**
 * Get the connectOptions used to connect this client to the server
 * @return The connectOptions used to connect the client to the server
 */
public MqttConnectOptions getConnectionOptions()
{
  return conOpt;
}

/**
 * Register a {@link PropertyChangeListener} to this object
 * @param listener the listener to register
 */
public void registerChangeListener(PropertyChangeListener listener)
{
  listeners.add(listener);
}

/**
 * Remove a registered {@link PropertyChangeListener}
 * @param listener A reference to the listener to remove
 */
public void removeChangeListener(PropertyChangeListener listener)
{
  if (listener != null) {
    listeners.remove(listener);
  }
}

/**
 * Notify {@link PropertyChangeListener} objects that the object has been updated
 * @param propertyChangeEvent
 */
private void notifyListeners(PropertyChangeEvent propertyChangeEvent)
{
  for (PropertyChangeListener listener : listeners)
  {
    listener.propertyChange(propertyChangeEvent);
  }
}

/**
 * Gets the port that this connection connects to.
 * @return port that this connection connects to
 */
public int getPort() {
  return port;
}

/**
 * Determines if the connection is secured using SSL, returning a C style integer value
 * @return 1 if SSL secured 0 if plain text
 */
public int isSSL() {
  return sslConnection ? 1 : 0;
}

/**
 * Assign a persistence ID to this object
 * @param id the persistence id to assign
 */
public void assignPersistenceId(long id) {
  persistenceId = id;
}

/**
 * Returns the persistence ID assigned to this object
 * @return the persistence ID assigned to this object
```

```
   */
  public long persistenceId() {
    return persistenceId;
  }
}
```

**Connection Details**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.app.ActionBar;
import android.app.FragmentTransaction;
import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.ViewPager;
import android.view.Menu;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapView;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.util.ArrayList;

/**
```

```
 * The connection details activity operates the fragments that make up the
 * connection details screen.
 * <p>
 * The fragments which this FragmentActivity uses are
 * <ul>
 * <li>{@link HistoryFragment}
 * <li>{@link PublishFragment}
 * <li>{@link SubscribeFragment}
 * </ul>
 *
 */
@SuppressWarnings("ALL")
public class ConnectionDetails extends FragmentActivity implements ActionBar.TabListener {

  /**
   * {@link SectionsPagerAdapter} that is used to get pages to display
   */
  SectionsPagerAdapter sectionsPagerAdapter;
  /**
   * {@link ViewPager} object allows pages to be flipped left and right
   */
  ViewPager viewPager;

  /** The currently selected tab **/
  private int selected = 0;

  /**
   * The handle to the {@link Connection} which holds the data for the client
   * selected
   **/
  private String clientHandle = null;

  /** This instance of <code>ConnectionDetails</code> **/
  private final ConnectionDetails connectionDetails = this;

  /**
   * The instance of {@link Connection} that the <code>clientHandle</code>
   * represents
   **/
  private Connection connection = null;

  /**
   * The {@link ChangeListener} this object is using for the connection
   * updates
   **/
  private ChangeListener changeListener = null;


  MapView gMap;
  GoogleMap map;

/*  private OnMapReadyCallback onMapReadyCallback = null;

  GoogleMap gMap;*/

  /**
   * @see android.support.v4.app.FragmentActivity#onCreate(android.os.Bundle)
   */
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    clientHandle = getIntent().getStringExtra("handle");
    setContentView(R.layout.activity_connection_details);
    // Create the adapter that will return a fragment for each of the pages
    sectionsPagerAdapter = new SectionsPagerAdapter(
        getSupportFragmentManager());

    // Set up the action bar for tab navigation
    final ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

    // add the sectionsPagerAdapter
    viewPager = (ViewPager) findViewById(R.id.pager);
    viewPager.setAdapter(sectionsPagerAdapter);

    // add the OnClickListener
/*    AdapterView.OnItemClickListener onItemClickListener = new
AdapterView.OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String key = GetHistoryID(position);
        if (ActivityConstants.Map ==1){
          SetMarker(key);
        }
      }
    };*/

    viewPager
        .setOnPageChangeListener(new ViewPager.SimpleOnPageChangeListener() {

      @Override
      public void onPageSelected(int position) {
        // select the tab that represents the current page
        actionBar.setSelectedNavigationItem(position);

      }
    });

    // Create the tabs for the screen
    for (int i = 0; i < sectionsPagerAdapter.getCount(); i++) {
      ActionBar.Tab tab = actionBar.newTab();
      tab.setText(sectionsPagerAdapter.getPageTitle(i));
      tab.setTabListener(this);
      actionBar.addTab(tab);
    }

    connection = Connections.getInstance(this).getConnection(clientHandle);
    changeListener = new ChangeListener();
    connection.registerChangeListener(changeListener);
  }

  @Override
  protected void onDestroy() {
    connection.removeChangeListener(null);
    super.onDestroy();
  }

  /**
   * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
   */
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    int menuID;
    Integer button = null;
    boolean connected = Connections.getInstance(this)
        .getConnection(clientHandle).isConnected();

    // Select the correct action bar menu to display based on the
  // connectionStatus and which tab is selected
  if (connected) {

    switch (selected) {
      case 0 : // history view
        menuID = R.menu.activity_connection_details;
        break;
      case 1 : // subscribe view
        menuID = R.menu.activity_connection_details;
        break;
      case 2 : // publish view
        menuID = R.menu.activity_publish;
        button = R.id.publish;
        break;
      default :
        menuID = R.menu.activity_connection_details;
        break;
    }
  }
  else {
    switch (selected) {
      case 0 : // history view
        menuID = R.menu.activity_connection_details_disconnected;
        break;
      case 1 : // subscribe view
        menuID = R.menu.activity_subscribe_disconnected;
        break;
      case 2 : // publish view
        menuID = R.menu.activity_publish_disconnected;
        button = R.id.publish;
        break;
      default :
        menuID = R.menu.activity_connection_details_disconnected;
        break;
    }
  }
  // inflate the menu selected
  getMenuInflater().inflate(menuID, menu);
  Listener listener = new Listener(this, clientHandle);
  // add listeners
  if (button != null) {
    // add listeners
    menu.findItem(button).setOnMenuItemClickListener(listener);
```

```
    if (!Connections.getInstance(this).getConnection(clientHandle)
        .isConnected()) {
      menu.findItem(button).setEnabled(false);
    }
  }
  // add the listener to the disconnect or connect menu option
  if (connected) {
    menu.findItem(R.id.disconnect).setOnMenuItemClickListener(listener);
  }
  else {
    menu.findItem(R.id.connectMenuOption).setOnMenuItemClickListener(
        listener);
  }

  return true;
}

/**
 * @see android.app.ActionBar.TabListener#onTabUnselected(android.app.ActionBar.Tab,
 *      android.app.FragmentTransaction)
 */
@Override
public void onTabUnselected(ActionBar.Tab tab,
    FragmentTransaction fragmentTransaction) {
  // Don't need to do anything when a tab is unselected
}

/**
 * @see android.app.ActionBar.TabListener#onTabSelected(android.app.ActionBar.Tab,
 *      android.app.FragmentTransaction)
 */
@Override
public void onTabSelected(ActionBar.Tab tab,
    FragmentTransaction fragmentTransaction) {
  // When the given tab is selected, switch to the corresponding page in
  // the ViewPager.
  viewPager.setCurrentItem(tab.getPosition());
  selected = tab.getPosition();
  // invalidate the options menu so it can be updated
  invalidateOptionsMenu();
  // history fragment is at position zero so get this then refresh its
  // view
  ((HistoryFragment) sectionsPagerAdapter.getItem(0)).refresh();
```

```
}

/**
 * @see android.app.ActionBar.TabListener#onTabReselected(android.app.ActionBar.Tab,
 *      android.app.FragmentTransaction)
 */
@Override
public void onTabReselected(ActionBar.Tab tab,
    FragmentTransaction fragmentTransaction) {
  // Don't need to do anything when the tab is reselected
}

/*@Override
public void onMapReady(GoogleMap gMap) {
  ActivityConstants.Map=1;
}*/



/**
 * Provides the Activity with the pages to display for each tab
 *
 */
public class SectionsPagerAdapter extends FragmentPagerAdapter {

  // Stores the instances of the pages
  private ArrayList<Fragment> fragments = null;

  /**
   * Only Constructor, requires a the activity's fragment managers
   *
   * @param fragmentManager
   */

  public SectionsPagerAdapter(FragmentManager fragmentManager) {
    super(fragmentManager);
    fragments = new ArrayList<Fragment>();
    // create the history view, passes the client handle as an argument
    // through a bundle
    Fragment fragment = new HistoryFragment();
    Bundle args = new Bundle();
    args.putString("handle", getIntent().getStringExtra("handle"));
    fragment.setArguments(args);
    // add all the fragments for the display to the fragments list
    fragments.add(fragment);
    fragments.add(new MapFrag());
    fragments.add(new NotificationFragment());
  }

  /**
   * @see android.support.v4.app.FragmentPagerAdapter#getItem(int)
   */
  @Override
  public Fragment getItem(int position) {
    return fragments.get(position);
  }

  /**
   * @see android.support.v4.view.PagerAdapter#getCount()
   */
  @Override
  public int getCount() {
    return fragments.size();
  }

  /**
   *
   * @see FragmentPagerAdapter#getPageTitle(int)
   */
  @Override
  public CharSequence getPageTitle(int position) {
    switch (position) {
      case 0 :
        return getString(R.string.history).toUpperCase();
      case 1 :
        return getString(R.string.title_maps_activity).toUpperCase();
      case 2 :
        return getString(R.string.title_activity_notification).toUpperCase();
    }
    // return null if there is no title matching the position
    return null;
  }

}

/**
```

```
   * <code>ChangeListener</code> updates the UI when the {@link Connection}
   * object it is associated with updates
   *
   */
  private class ChangeListener implements PropertyChangeListener {

    /**
     * @see
java.beans.PropertyChangeListener#propertyChange(java.beans.PropertyChangeEvent)
     */
    @Override
    public void propertyChange(PropertyChangeEvent event) {
      // connection object has change refresh the UI

      connectionDetails.runOnUiThread(new Runnable() {

        @Override
        public void run() {
          connectionDetails.invalidateOptionsMenu();
          ((HistoryFragment) connectionDetails.sectionsPagerAdapter
              .getItem(0)).refresh();

        }
      });

    }
  }



  public void SetMarker (String key){
    String name = GetName(key);
    Double[] Latlng = GetGps(key);
    final LatLng Item = new LatLng(Latlng[0], Latlng[1]);
    gMap = (MapView) findViewById(R.id.mapfrag);
    map = gMap.getMap();
    map.addMarker(new MarkerOptions().title(name).position(Item));
  }
  public Double[] GetGps(String id){
    SharedPreferences sharedPreferences = getSharedPreferences("objectlocation",
Context.MODE_PRIVATE);
    String Location = sharedPreferences.getString(id, null);
    return processLocation(Location);
  }

  public String GetHistoryID (int position){
    SharedPreferences sharedPreferences = getSharedPreferences("historydetails",
Context.MODE_ENABLE_WRITE_AHEAD_LOGGING);
    return sharedPreferences.getString(Integer.toString(position), null);
  }

  public Double[] processLocation (String location){
    String[] Location = location.split(",");
    Double[] LatLng = new Double[2];
    LatLng[0] = MintoDec(Location[0]);
    LatLng[1] = -MintoDec(Location[1]);
    return LatLng;
  }

  public Double MintoDec(String location){
    double a = Double.parseDouble(location);
    double d = (int)a / 100;
    a -= d*100;
    return d +(a/60);
  }

  public String GetName(String id){

    SharedPreferences sharedPreferences =getSharedPreferences("objectdetails",
Context.MODE_ENABLE_WRITE_AHEAD_LOGGING);
    return sharedPreferences.getString(id,null);
  }
}
```

**Connections**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.content.Context;

import org.eclipse.paho.android.service.MqttAndroidClient;

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

/**
 * <code>Connections</code> is a singleton class which stores all the connection objects
 * in one central place so they can be passed between activities using a client
 * handle
 *
 */
public class Connections {

  /** Singleton instance of <code>Connections</code>**/
  private static Connections instance = null;

  /** List of {@link Connection} objects**/
  private HashMap<String, Connection> connections = null;

  /** {@link Persistence} object used to save, delete and restore connections**/
  private Persistence persistence = null;

  /**
   * Create a Connections object
   * @param context Applications context
   */
  private Connections(Context context)
  {
    connections = new HashMap<String, Connection>();

    //attempt to restore state
    persistence = new Persistence(context);
    try {
      List<Connection> l = persistence.restoreConnections(context);
      for (Connection c : l) {
        connections.put(c.handle(), c);
      }
    }
    catch (PersistenceException e) {
      e.printStackTrace();
    }

  }

  /**
   * Returns an already initialised instance of <code>Connections</code>, if Connections has
yet to be created, it will
   * create and return that instance
   * @param context The applications context used to create the <code>Connections</code>
object if it is not already initialised
   * @return Connections instance
   */
  public synchronized static Connections getInstance(Context context)
  {
    if (instance == null) {
      instance = new Connections(context);
    }

    return instance;
  }

  /**
   * Finds and returns a connection object that the given client handle points to
   * @param handle The handle to the <code>Connection</code> to return
   * @return a connection associated with the client handle, <code>null</code> if one is not
found
   */
  public Connection getConnection(String handle)
  {

    return connections.get(handle);
  }

  /**
   * Adds a <code>Connection</code> object to the collection of connections associated with
this object
   * @param connection connection to add
   */
  public void addConnection(Connection connection)
```

```
{
  connections.put(connection.handle(), connection);
  try {
    persistence.persistConnection(connection);
  }
  catch (PersistenceException e)
  {
    //error persisting well lets just swallow this
    e.printStackTrace();
  }
}

/**
 * Create a fully initialised <code>MqttAndroidClient</code> for the parameters given
 * @param context The Applications context
 * @param serverURI The ServerURI to connect to
 * @param clientId The clientId for this client
 * @return new instance of MqttAndroidClient
 */
public MqttAndroidClient createClient(Context context, String serverURI, String clientId)
{
  MqttAndroidClient client = new MqttAndroidClient(context, serverURI, clientId);
  return client;
}

/**
 * Get all the connections associated with this <code>Connections</code> object.
 * @return <code>Map</code> of connections
 */
public Map<String, Connection> getConnections()
{
  return connections;
}

/**
 * Removes a connection from the map of connections
 * @param connection connection to be removed
 */
public void removeConnection(Connection connection) {
  connections.remove(connection.handle());
  persistence.deleteConnection(connection);
}
```

```
}
```

**History Fragment**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.content.Context;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.text.Spanned;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapView;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

/**
 * This fragment displays the history information for a client
 *
 */
public class HistoryFragment extends ListFragment {

  /**
   * Client handle to a {@link Connection} object
   **/
  String clientHandle = null;
  /**
   * {@link ArrayAdapter} to display the formatted text
   **/
  ArrayAdapter<Spanned> arrayAdapter = null;

  GoogleMap map;
  MapView gMap;

  @Override
  public void onViewCreated(View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

/*    final ListView histlist = getListView();
    histlist.setOnItemClickListener(new AdapterView.OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String key = GetHistoryID(position);
        GoogleMap gmap;
        if (key != null) {
          String name = GetName(key);
          Double[] Latlng = GetGps(key);
          final LatLng Item = new LatLng(Latlng[0], Latlng[1]);
          gmap = ((SupportMapFragment)
getFragmentManager().findFragmentById(R.id.map)).getMapAsync(OnMapReadyCallback());
          Marker item = gmap.addMarker(new MarkerOptions().title(name).position(Item));
        }
      }
    });*/

  }

  /**
   * @see ListFragment#onCreate(Bundle)
   */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //Pull history information out of bundle


    clientHandle = getArguments().getString("handle");
    Connection connection = Connections.getInstance(getActivity()).getConnection(clientHandle);
    Spanned[] history = connection.history();

    //Initialise the arrayAdapter, view and add data
    arrayAdapter = new ArrayAdapter<Spanned>(getActivity(), R.layout.list_view_text_view);

    arrayAdapter.addAll(history);
    setListAdapter(arrayAdapter);


  }

  @Override
  public void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
      String message = l.getItemAtPosition(position).toString();
      String[] relevant = message.split(" ");
      String key = RightId(relevant[1]);
      SetMarker(key);
  }

  public String RightId(String id){
    Double[] trial = GetGps(GetName(id));
    if (trial == null){
      return id;
    }
    else{
      return GetName(id);
    }
  }

/*  * Code for getting location of scanned objects
   *  Kimbo-Made*/
  public Double[] GetGps(String id){
    SharedPreferences sharedPreferences =
getContext().getSharedPreferences("objectlocation", Context.MODE_PRIVATE);
    String Location = sharedPreferences.getString(id, null);
if(Location != null) {
  return processLocation(Location);
}
    else return null;
  }
```

```
/*  public String GetHistoryID (int position){
    SharedPreferences sharedPreferences = getContext().getSharedPreferences("historydetails",
Context.MODE_ENABLE_WRITE_AHEAD_LOGGING);
    return sharedPreferences.getString(Integer.toString(position), null);
  }*/

  public Double[] processLocation (String location){
    if(location!=null){
    String[] Location = location.split(",");
    Double[] LatLng = new Double[2];
    LatLng[0] = MintoDec(Location[0]);
    LatLng[1] = -MintoDec(Location[1]);
    return LatLng;}
    else return null;
  }

  public Double MintoDec(String location){
    double a = Double.parseDouble(location);
    double d = (int)a / 100;
    a -= d*100;
    return d +(a/60);
  }

  public String GetName(String id){

    SharedPreferences sharedPreferences = getContext().getSharedPreferences("objectdetails",
Context.MODE_ENABLE_WRITE_AHEAD_LOGGING);
    return sharedPreferences.getString(id,null);
  }

  public void SetMarker (String key){
    String name = GetName(key);
    Double[] Latlng = GetGps(key);
    if(Latlng!=null){
    final LatLng Item = new LatLng(Latlng[0], Latlng[1]);
    gMap = (MapView) getActivity().findViewById(R.id.mapfrag);
    map = gMap.getMap();
    map.clear();

      map.addMarker(new MarkerOptions().title(name).position(Item));
    //}
    CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(Item, 17);
    map.animateCamera(cameraUpdate);}
  }


  /**
   * Updates the data displayed to match the current history
   */
  public void refresh() {
    if (arrayAdapter != null) {
      arrayAdapter.clear();

arrayAdapter.addAll(Connections.getInstance(getActivity()).getConnection(clientHandle).history(
));
      arrayAdapter.notifyDataSetChanged();
    }
  }
}
```

**Listener**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.util.Log;
import android.view.MenuItem;
import android.view.MenuItem.OnMenuItemClickListener;
import android.widget.EditText;
import android.widget.Toast;

import org.eclipse.paho.android.service.MqttAndroidClient;
import org.eclipse.paho.android.service.sample.ActionListener.Action;
import org.eclipse.paho.android.service.sample.Connection.ConnectionStatus;
import org.eclipse.paho.client.mqttv3.IMqttToken;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttSecurityException;

import java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.Map.Entry;
import java.util.logging.LogManager;

/**
 * Deals with actions performed in the {@link ClientConnections} activity
 * and the {@link ConnectionDetails} activity and associated fragments
 *
 */
public class Listener implements OnMenuItemClickListener{

  /** The handle to a {@link Connection} object which contains the {@link MqttAndroidClient}
associated with this object **/
  private String clientHandle = null;

  /** {@link ConnectionDetails} reference used to perform some actions**/
  private ConnectionDetails connectionDetails = null;
  /** {@link ClientConnections} reference used to perform some actions**/
  private ClientConnections clientConnections = null;
  /** {@link Context} used to load and format strings **/
  private Context context = null;

  /** Whether Paho is logging is enabled**/
  static boolean logging = false;

  /**
   * Constructs a listener object for use with {@link ConnectionDetails} activity and
   * associated fragments.
   * @param connectionDetails The instance of {@link ConnectionDetails}
   * @param clientHandle The handle to the client that the actions are to be performed on
   */
  public Listener(ConnectionDetails connectionDetails, String clientHandle)
  {
    this.connectionDetails = connectionDetails;
    this.clientHandle = clientHandle;
    context = connectionDetails;
```

```
}

/**
 * Constructs a listener object for use with {@link ClientConnections} activity.
 * @param clientConnections The instance of {@link ClientConnections}
 */
public Listener(ClientConnections clientConnections) {
  this.clientConnections = clientConnections;
  context = clientConnections;
}


/**
 * Perform the needed action required based on the button that
 * the user has clicked.
 *
 * @param item The menu item that was clicked
 * @return If there is anymore processing to be done
 *
 */

@Override
public boolean onMenuItemClick(MenuItem item) {

  int id = item.getItemId();

  switch (id)
  {
    case R.id.publish :
      saveName();
      saveId();
      Notify.toast(context, "You renamed your item", Toast.LENGTH_LONG);
      break;
    case R.id.subscribe :
      subscribe();
      break;
    case R.id.newConnection :
      createAndConnect();
      break;
    case R.id.disconnect :
      disconnect();
      break;
    case R.id.connectMenuOption :
        reconnect();
        //subscribe();
        break;
      case R.id.startLogging :
        enablePahoLogging();
        break;
      case R.id.endLogging :
        disablePahoLogging();
        break;
    }

    return false;
  }

  /**
   * Reconnect the selected client
   */
  private void reconnect() {


Connections.getInstance(context).getConnection(clientHandle).changeConnectionStatus(Conne
ctionStatus.CONNECTING);

    Connection c = Connections.getInstance(context).getConnection(clientHandle);
    try {
      IMqttToken connect = c.getClient().connect(c.getConnectionOptions(), null, new
ActionListener(context, Action.CONNECT, clientHandle, null));
    }
    catch (MqttSecurityException e) {
      Log.e(this.getClass().getCanonicalName(), "Failed to reconnect the client with the handle " +
clientHandle, e);
      c.addAction("Client failed to connect");
    }
    catch (MqttException e) {
      Log.e(this.getClass().getCanonicalName(), "Failed to reconnect the client with the handle " +
clientHandle, e);
      c.addAction("Client failed to connect");
    }

  }

  /**
   * Disconnect the client
   */
  private void disconnect() {

    Connection c = Connections.getInstance(context).getConnection(clientHandle);

    //if the client is not connected, process the disconnect
    if (!c.isConnected()) {
      return;
    }

    try {
      c.getClient().disconnect(null, new ActionListener(context, Action.DISCONNECT,
clientHandle, null));
      c.changeConnectionStatus(ConnectionStatus.DISCONNECTING);
    }
    catch (MqttException e) {
      Log.e(this.getClass().getCanonicalName(), "Failed to disconnect the client with the handle "
+ clientHandle, e);
      c.addAction("Client failed to disconnect");
    }

  }

  /**
   * Subscribe to a topic that the user has specified
   */
  private void subscribe()
  {
    String topic = "trakPak";//((EditText)
connectionDetails.findViewById(R.id.topic)).getText().toString();
    //((EditText) connectionDetails.findViewById(R.id.topic)).getText().clear();

    //RadioGroup radio = (RadioGroup) connectionDetails.findViewById(R.id.qosSubRadio);
    //int checked = radio.getCheckedRadioButtonId();
    int qos = ActivityConstants.defaultQos;

   // switch (checked) {
      //case R.id.qos0 :
        //qos = 1;
        //break;
     //case R.id.qos1 :
        ////qos = 1;
        //break;
      //case R.id.qos2 :
        //qos = 1;
        //break;
    //}

    try {
      String[] topics = new String[1];
      topics[0] = topic;
      Connections.getInstance(context).getConnection(clientHandle).getClient().subscribe(topic,
qos, null, new ActionListener(context, Action.SUBSCRIBE, clientHandle, topic));
    }
    catch (MqttSecurityException e) {
      Log.e(this.getClass().getCanonicalName(), "Failed to subscribe to" + topic + " the client with
the handle " + clientHandle, e);
    }
    catch (MqttException e) {
      Log.e(this.getClass().getCanonicalName(), "Failed to subscribe to" + topic + " the client with
the handle " + clientHandle, e);
    }
  }

  private void saveName()
  {
    String key = ((EditText)
connectionDetails.findViewById(R.id.NumberText)).getText().toString();
    String name =
((EditText)connectionDetails.findViewById(R.id.NameText)).getText().toString();
    SharedPreferences ObjectDetails = context.getSharedPreferences("objectdetails",
Context.MODE_ENABLE_WRITE_AHEAD_LOGGING);
    SharedPreferences.Editor editor = ObjectDetails.edit();
    editor.putString(key,name);
    editor.apply();
  }

  private void saveId()
  {
    String key = ((EditText)
connectionDetails.findViewById(R.id.NumberText)).getText().toString();
    String name =
((EditText)connectionDetails.findViewById(R.id.NameText)).getText().toString();
    SharedPreferences ObjectDetails = context.getSharedPreferences("objectdetails",
Context.MODE_ENABLE_WRITE_AHEAD_LOGGING);
    SharedPreferences.Editor editor = ObjectDetails.edit();
    editor.putString(name,key);
    editor.apply();
  }
  /**
   * Publish the message the user has specified
   */
 /* private void publish()
  {
    String topic = "InTopic";//((EditText) connectionDetails.findViewById(R.id.lastWillTopic))
        //.getText().toString();

    ((EditText) connectionDetails.findViewById(R.id.lastWillTopic)).getText().clear();

    String message = ((EditText) connectionDetails.findViewById(R.id.lastWill)).getText()
        .toString();

    ((EditText) connectionDetails.findViewById(R.id.lastWill)).getText().clear();

    //RadioGroup radio = (RadioGroup) connectionDetails.findViewById(R.id.qosRadio);
    //int checked = radio.getCheckedRadioButtonId();
    int qos = ActivityConstants.defaultQos;

    //switch (checked) {
      //case R.id.qos0 :
       // qos = 1;
      //  break;
     // case R.id.qos1 :
       // qos = 1;
      //  break;
    //  case R.id.qos2 :
      //  qos = 1;
      //  break;
    //}

    boolean retained = ((CheckBox) connectionDetails.findViewById(R.id.retained))
       .isChecked();

    String[] args = new String[2];
    args[0] = message;
    args[1] = topic+";qos:"+qos+";retained:"+retained;

    try {
      Connections.getInstance(context).getConnection(clientHandle).getClient()
```

```
        .publish(topic, message.getBytes(), qos, retained, null, new ActionListener(context,
Action.PUBLISH, clientHandle, args));
    }
    catch (MqttSecurityException e) {
      Log.e(this.getClass().getCanonicalName(), "Failed to publish a messged from the client with
the handle " + clientHandle, e);
    }
    catch (MqttException e) {
      Log.e(this.getClass().getCanonicalName(), "Failed to publish a messged from the client with
the handle " + clientHandle, e);
    }

  }*/

  /**
   * Create a new client and connect
   */
  private void createAndConnect()
  {
    Intent createConnection;

    //start a new activity to gather information for a new connection
    createConnection = new Intent();
    createConnection.setClassName(
        clientConnections.getApplicationContext(),
        "org.eclipse.paho.android.service.sample.NewConnection");

    clientConnections.startActivityForResult(createConnection,
        ActivityConstants.connect);
  }

  /**
   * Enables logging in the Paho MQTT client
   */
  private void enablePahoLogging() {

    try {
      InputStream logPropStream =
context.getResources().openRawResource(R.raw.jsr47android);
      LogManager.getLogManager().readConfiguration(logPropStream);
      logging = true;
      HashMap<String, Connection> connections =
(HashMap<String,Connection>)Connections.getInstance(context).getConnections();
      if(!connections.isEmpty()){
              Entry<String, Connection> entry = connections.entrySet().iterator().next();
              Connection connection = (Connection)entry.getValue();
              connection.getClient().setTraceEnabled(true);
              //change menu state.
              clientConnections.invalidateOptionsMenu();

//Connections.getInstance(context).getConnection(clientHandle).getClient().setTraceEnabled(tr
ue);
      }else{
              Log.i("SampleListener","No connection to enable log in service");
      }
    }
    catch (IOException e) {
      Log.e("MqttAndroidClient",
          "Error reading logging parameters", e);
    }

  }

  /**
   * Disables logging in the Paho MQTT client
   */
  private void disablePahoLogging() {
    LogManager.getLogManager().reset();
    logging = false;

    HashMap<String, Connection> connections =
(HashMap<String,Connection>)Connections.getInstance(context).getConnections();
    if(!connections.isEmpty()){
              Entry<String, Connection> entry = connections.entrySet().iterator().next();
              Connection connection = (Connection)entry.getValue();
              connection.getClient().setTraceEnabled(false);
              //change menu state.
              clientConnections.invalidateOptionsMenu();
    }else{
              Log.i("SampleListener","No connection to disable log in service");
    }
    clientConnections.invalidateOptionsMenu();
  }
```

```
}
```

**Map Frag**

```
package org.eclipse.paho.android.service.sample;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.google.android.gms.maps.CameraUpdate;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapView;
import com.google.android.gms.maps.MapsInitializer;
import com.google.android.gms.maps.model.LatLng;

/**
 * Created by Michael on 4/27/2016.
 */
public class MapFrag extends android.support.v4.app.Fragment {
   MapView mapView;
   GoogleMap map;

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
       View v = inflater.inflate(R.layout.map_frag, container, false);

       // Gets the MapView from the XML layout and creates it
       mapView = (MapView) v.findViewById(R.id.mapfrag);
       mapView.onCreate(savedInstanceState);

       // Gets to GoogleMap from the MapView and does initialization stuff
       map = mapView.getMap();
       //map.getUiSettings().setMyLocationButtonEnabled(false);
       //map.setMyLocationEnabled(true);
       //map.addMarker(new MarkerOptions().position(new LatLng(41.702622,-86.238967)));


       // Needs to call MapsInitializer before doing any CameraUpdateFactory calls
       MapsInitializer.initialize(this.getActivity());
```

```
        // Updates the location and zoom of the MapView
        CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLngZoom(new
LatLng(41.702622,-86.238967), 17);
        map.animateCamera(cameraUpdate);

        return v;
    }

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

    @Override
    public void onDestroy() {
        super.onDestroy();
        mapView.onDestroy();
    }

    @Override
    public void onLowMemory() {
        super.onLowMemory();
        mapView.onLowMemory();
    }
}
```

**Mqtt Callback Handler**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import org.eclipse.paho.android.service.sample.Connection.ConnectionStatus;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttMessage;

import java.nio.charset.StandardCharsets;

/**
 * Handles call backs from the MQTT Client
 *
 */
public class MqttCallbackHandler extends Activity implements MqttCallback {

  /** {@link Context} for the application used to format and import external strings**/
  private Context context;
  /** Client handle to reference the connection that this handler is attached to**/
  private String clientHandle;

  /**
   * Creates an <code>MqttCallbackHandler</code> object
   * @param context The application's context
   * @param clientHandle The handle to a {@link Connection} object
   */
  public MqttCallbackHandler(Context context, String clientHandle)
  {
    this.context = context;
    this.clientHandle = clientHandle;
  }


  public void saveGps(String key, String value) {
    SharedPreferences ObjectLocation =
context.getSharedPreferences("objectlocation",context.MODE_PRIVATE);
    Editor editor = ObjectLocation.edit();
    editor.putString(key, value);
    editor.apply();
  }


  /** Code for obtaining scanned objects name
   *  Kimbo-Made
   */


  /** Code for renaming scanned objects
   *  Kimbo-Made
   */

  public String GetName(String id){

    SharedPreferences sharedPreferences =
context.getSharedPreferences("objectdetails",Context.MODE_ENABLE_WRITE_AHEAD_LOG
GING);
    return sharedPreferences.getString(id, null);
  }

  /** Code for parsing the payload
   *  Kimbo-Made
   */

  public String[] Transcribe(String string)
  {
    return string.split(",");
  }

  public String TimeShift(String time){
    double t = Double.parseDouble(time);
    int h = (int)t/10000;
    int m = (int)t/100 -h*100;
    h = h-4;
    if(h==-1){
      h = 23;
    }
    else if(h==-2){
      h = 22;
    }
    else if(h==-3){
      h = 21;
    }
    else if(h==-4){
      h = 20;
    }
    return h + ":" + m;
  }


  /**
   * @see org.eclipse.paho.client.mqttv3.MqttCallback#connectionLost(java.lang.Throwable)
   */
  @Override
  public void connectionLost(Throwable cause) {
//          cause.printStackTrace();
    if (cause != null) {
      Connection c = Connections.getInstance(context).getConnection(clientHandle);
      c.addAction("Connection Lost");
      c.changeConnectionStatus(ConnectionStatus.DISCONNECTED);

      //format string to use a notification text
      Object[] args = new Object[2];
      args[0] = c.getId();
      args[1] = c.getHostName();

      String message = context.getString(R.string.connection_lost, args);

      //build intent
      Intent intent = new Intent();
      intent.setClassName(context,
"org.eclipse.paho.android.service.sample.ConnectionDetails");
      intent.putExtra("handle", clientHandle);

      //notify the user
      Notify.notifcation(context, message, intent, R.string.notifyTitle_connectionLost);
    }
  }

  /**
   * @see org.eclipse.paho.client.mqttv3.MqttCallback#messageArrived(java.lang.String,
org.eclipse.paho.client.mqttv3.MqttMessage)
   */

 public void setNumText (String name) {
   EditText editText = (EditText) findViewById(R.id.NumberText);
   editText.setText(name, TextView.BufferType.EDITABLE);
 }


  @Override
  public void messageArrived(String topic, MqttMessage message) throws Exception {

    //Get connection object associated with this object
    Connection c = Connections.getInstance(context).getConnection(clientHandle);

    //parse payload string
    String str = new String(message.getPayload(), StandardCharsets.UTF_8);
    String[] parse = Transcribe(str);
    String name = parse[0];
    String Time = TimeShift(parse[2]);
    String gps = parse[3] + "," + parse[5];

    saveGps(name, gps);

    //create arguments to format message arrived notifcation string
    String[] args = new String[3];

    if (GetName(name)!=null){
      args[0] = GetName(name);}
    else{
      args[0] = name;
      //setNumText(name);
      Notify.toast(context,"Please name your item", Toast.LENGTH_LONG);
    }

    args[1] = Time;

    //get the string from strings.xml and format
    String messageString = context.getString(R.string.messageRecieved, (Object[]) args);

    //create intent to start activity
    Intent intent = new Intent();
    intent.setClassName(context, "org.eclipse.paho.android.service.sample.ConnectionDetails");
    intent.putExtra("handle", clientHandle);

    //format string args
    Object[] notifyArgs = new String[4];
    notifyArgs[0] = c.getId();

    if (GetName(name)==null){
      notifyArgs[1] = name;}
    else{
      notifyArgs[1] = GetName(name);}

    notifyArgs[2] = Time;
    notifyArgs[3] = topic;

    //notify the user
    int messageID = Notify.GetMessageID();

    //saveId(Integer.toString(messageID),name);

    Notify.notifcation(context, context.getString(R.string.notification, notifyArgs), intent,
R.string.notifyTitle);


    //update client history
    c.addAction(messageString);
  }

  /**
   * @see
org.eclipse.paho.client.mqttv3.MqttCallback#deliveryComplete(org.eclipse.paho.client.mqttv3.IM
qttDeliveryToken)
   */
  @Override
  public void deliveryComplete(IMqttDeliveryToken token) {
    // Do nothing
  }
}
```

**Mqtt Trace Callback**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
```

```
package org.eclipse.paho.android.service.sample;

import android.util.Log;

import org.eclipse.paho.android.service.MqttTraceHandler;

public class MqttTraceCallback implements MqttTraceHandler {

	public void traceDebug(java.lang.String arg0, java.lang.String arg1) {
		Log.i(arg0, arg1);
	};

	public void traceError(java.lang.String arg0, java.lang.String arg1) {
		Log.e(arg0, arg1);
	};

	public void traceException(java.lang.String arg0, java.lang.String arg1,
			java.lang.Exception arg2) {
		Log.e(arg0, arg1, arg2);
	};

}
```

**New Connection**

```
package org.eclipse.paho.android.service.sample;
```

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MenuItem.OnMenuItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.EditText;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

/**
 * Handles collection of user information to create a new MQTT Client
 *
 */
public class NewConnection extends Activity {

  /** {@link Bundle} which holds data from activities launched from this activity **/
  private Bundle result = null;

  /**
   * @see android.app.Activity#onCreate(android.os.Bundle)
   */
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_new_connection);

    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1);
    adapter.addAll(readHosts());
    //AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id.serverURI);
    //textView.setAdapter(adapter);

    //load auto compete options
  }

  /**
   * @see android.app.Activity#onCreateOptionsMenu(android.view.Menu)
   */
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_new_connection, menu);
    OnMenuItemClickListener listener = new Listener(this);
    menu.findItem(R.id.connectAction).setOnMenuItemClickListener(listener);
    menu.findItem(R.id.advanced).setOnMenuItemClickListener(listener);

    return true;
  }

  /**
   * @see android.app.Activity#onOptionsItemSelected(android.view.MenuItem)
   */
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
      case android.R.id.home :
        NavUtils.navigateUpFromSameTask(this);
        return true;
    }
    return super.onOptionsItemSelected(item);
  }

  /**
   * @see android.app.Activity#onActivityResult(int, int, android.content.Intent)
   */
  @Override
  protected void onActivityResult(int requestCode, int resultCode,
      Intent intent) {

    if (resultCode == RESULT_CANCELED) {
      return;
    }

    result = intent.getExtras();

  }
```

```
  /**
   * Handles action bar actions
   *
   */
  private class Listener implements OnMenuItemClickListener {

    //used for starting activities
    private NewConnection newConnection = null;

    public Listener(NewConnection newConnection)
    {
      this.newConnection = newConnection;
    }

    /**
     * @see
android.view.MenuItem.OnMenuItemClickListener#onMenuItemClick(android.view.MenuItem)
     */
    @Override
    public boolean onMenuItemClick(MenuItem item) {
      {
        // this will only connect need to package up and sent back

        int id = item.getItemId();

        Intent dataBundle = new Intent();

        switch (id) {
          case R.id.connectAction :
            //extract client information
            String server = (ActivityConstants.server);  //((AutoCompleteTextView)
findViewById(R.id.serverURI))
              // .getText().toString();
            String port = ActivityConstants.port;//((EditText) findViewById(R.id.port))
               //.getText().toString();
            String clientId = ((EditText) findViewById(R.id.clientId))
               .getText().toString();

           /* if (server.equals(ActivityConstants.empty) || port.equals(ActivityConstants.empty) ||
clientId.equals(ActivityConstants.empty))
            {
              String notificationText = newConnection.getString(R.string.missingOptions);
              Notify.toast(newConnection, notificationText, Toast.LENGTH_LONG);
              return false;
            }*/

            boolean cleanSession = true;//((CheckBox)
findViewById(R.id.cleanSessionCheckBox)).isChecked();
            //persist server
            //persistServerURI(server);

            //put data into a bundle to be passed back to ClientConnections
            dataBundle.putExtra(ActivityConstants.server, server);
            dataBundle.putExtra(ActivityConstants.port, port);
            dataBundle.putExtra(ActivityConstants.clientId, clientId);
            dataBundle.putExtra(ActivityConstants.action, ActivityConstants.connect);
            dataBundle.putExtra(ActivityConstants.cleanSession, cleanSession);

            if (result == null) {
              // create a new bundle and put default advanced options into a bundle
              result = new Bundle();

              result.putString(ActivityConstants.message,
                  ActivityConstants.empty);
              result.putString(ActivityConstants.topic, ActivityConstants.empty);
              result.putInt(ActivityConstants.qos, ActivityConstants.defaultQos);
              result.putBoolean(ActivityConstants.retained,
                  ActivityConstants.defaultRetained);

              result.putString(ActivityConstants.username,
                  ActivityConstants.empty);
              result.putString(ActivityConstants.password,
                  ActivityConstants.empty);

              result.putInt(ActivityConstants.timeout,
                  ActivityConstants.defaultTimeOut);
              result.putInt(ActivityConstants.keepalive,
                  ActivityConstants.defaultKeepAlive);
              result.putBoolean(ActivityConstants.ssl,
                  ActivityConstants.defaultSsl);

            }
            //add result bundle to the data being returned to ClientConnections
            dataBundle.putExtras(result);

            setResult(RESULT_OK, dataBundle);
```

```
        newConnection.finish();
        break;
      case R.id.advanced :
        //start the advanced options activity
        dataBundle.setClassName(newConnection,
            "org.eclipse.paho.android.service.sample.Advanced");
        newConnection.startActivityForResult(dataBundle,
            ActivityConstants.advancedConnect);

        break;
    }
    return false;

  }

}

/**
 * Add a server URI to the persisted file
 *
 * @param serverURI the uri to store
 */
private void persistServerURI(String serverURI) {
  File fileDir = newConnection.getFilesDir();
  File presited = new File(fileDir, "hosts.txt");
  BufferedWriter bfw = null;
  try {
    bfw = new BufferedWriter(new FileWriter(presited));
    bfw.write(serverURI);
    bfw.newLine();
  }
  catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
  finally {
    try {
      if (bfw != null) {
        bfw.close();
      }
    }
    catch (IOException e) {
      // TODO Auto-generated catch block
```

```
        e.printStackTrace();
      }
    }
  }

}

/**
 * Read persisted hosts
 * @return The hosts contained in the persisted file
 */
private String[] readHosts() {
  File fileDir = getFilesDir();
  File persisted = new File(fileDir, "hosts.txt");
  if (!persisted.exists()) {
    return new String[0];
  }
  ArrayList<String> hosts = new ArrayList<String>();
  BufferedReader br = null;
  try {
    br = new BufferedReader(new FileReader(persisted));
    String line = null;
    line = br.readLine();
    while (line != null) {
      hosts.add(line);
      line = br.readLine();
    }
  }
  catch (IOException e) {
    e.printStackTrace();
  }
  finally {
    try {
      if (br != null) {
        br.close();
      }
    }
    catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
  }
```

```
    return hosts.toArray(new String[hosts.size()]);

  }
}
```

**Notification**

```
package org.eclipse.paho.android.service.sample;

/**
 * Created by Michael on 4/9/2016.
 */

import android.app.Activity;
import android.os.Bundle;
import android.widget.EditText;

public class notification extends Activity {

    EditText nameText;
    EditText numText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.notification);
        nameText = (EditText) findViewById(R.id.NameText);
        numText = (EditText) findViewById(R.id.NumberText);
    }
    /*public void setNumText (String name){
        EditText editText = (EditText) findViewById(R.id.NumberText);
        editText.setText(name, TextView.BufferType.EDITABLE);
    }*/
}
```

**Notification Fragment**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

/**
 * Fragment for the publish message pane.
 *
 */
public class NotificationFragment extends Fragment {

  /**
   * @see Fragment#onCreateView(LayoutInflater, ViewGroup, Bundle)
   */
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
      Bundle savedInstanceState) {

    return LayoutInflater.from(getActivity()).inflate(R.layout.notification, null);
  }

}
```

**Notify**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
```

```
package org.eclipse.paho.android.service.sample;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat.Builder;
import android.widget.Toast;

import java.util.Calendar;

/**
 * Provides static methods for creating and showing notifications to the user.
 *
 */
public class Notify {

  /** Message ID Counter **/
  private static int MessageID = 0;

  /**
   * Displays a notification in the notification area of the UI
   * @param context Context from which to create the notification
   * @param messageString The string to display to the user as a message
   * @param intent The intent which will start the activity when the user clicks the notification
   * @param notificationTitle The resource reference to the notification title
   */
  static void notifcation(Context context, String messageString, Intent intent, int notificationTitle) {

    //Get the notification manage which we will use to display the notification
    String ns = Context.NOTIFICATION_SERVICE;
    NotificationManager mNotificationManager = (NotificationManager)
context.getSystemService(ns);

    Calendar.getInstance().getTime().toString();

    long when = System.currentTimeMillis();

    //get the notification title from the application's strings.xml file
    CharSequence contentTitle = context.getString(notificationTitle);

    //the message that will be displayed as the ticker
    String ticker = contentTitle + " " + messageString;

    //build the pending intent that will start the appropriate activity
    PendingIntent pendingIntent = PendingIntent.getActivity(context,
        ActivityConstants.showHistory, intent, 0);

    //build the notification
    Builder notificationCompat = new Builder(context);
    notificationCompat.setAutoCancel(true)
        .setContentTitle(contentTitle)
        .setContentIntent(pendingIntent)
        .setContentText(messageString)
        .setTicker(ticker)
        .setWhen(when)
        .setSmallIcon(R.drawable.ic_launcher);

    Notification notification = notificationCompat.build();
    //display the notification
    mNotificationManager.notify(MessageID, notification);
    MessageID++;

  }

  /**
   * Display a toast notification to the user
   * @param context Context from which to create a notification
   * @param text The text the toast should display
   * @param duration The amount of time for the toast to appear to the user
   */
  static void toast(Context context, CharSequence text, int duration) {
    Toast toast = Toast.makeText(context, text, duration);
    toast.show();
  }
    static int GetMessageID() {
        return MessageID;
    }

}
```

**Open File Dialog**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;


import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.Toast;

import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;

/**
 * Add SSL key file selector
 * @author foxxiang
 *
 */
public class OpenFileDialog {
	public static String tag = "OpenFileDialog";
	static final public String sRoot = "/";
	static final public String sParent = "..";
	static final public String sFolder = ".";
	static final public String sEmpty = "";
	static final private String sOnErrorMsg = "No rights to access!";
	/**
	 * Create a File Selector Dialog windows
	 * @param id Dialog Id
	 * @param context Context that the application is running in
	 * @param title The tile of File Selector Window
	 * @param callback A callback Bundle interface for data transport
	 * @param suffix The file name suffix. E.g.  .bks , .pem
	 * @param images The resource id for file icon
	 * @return The Dialog Window
	 */
	public static Dialog createDialog(int id, Context context, String title, CallbackBundle
callback, String suffix, Map<String, Integer> images){
		AlertDialog.Builder builder = new AlertDialog.Builder(context);
		builder.setView(new FileSelectView(context, id, callback, suffix, images));
		Dialog dialog = builder.create();
		//dialog.requestWindowFeature(Window.FEATURE_NO_TITLE);
		dialog.setTitle(title);
		return dialog;
	}

	/** The FileSelect View with OnItemClick Listener*/
	static class FileSelectView extends ListView implements OnItemClickListener{

		private CallbackBundle callback = null;
		private String path = sRoot;
		private List<Map<String, Object>> list = null;
		private int dialogid = 0;

		private String suffix = null;

		private Map<String, Integer> imagemap = null;

		/**
		 * Create the File Selector Dialog Window View
		 * @param id Dialog Id
		 * @param context Context that the application is running in
		 * @param title The tile of File Selector Window
		 * @param callback A callback Bundle interface for data transport
		 * @param suffix The file name suffix. E.g. .bks , .pem
		 * @param images The resource id for file icon
		 */
	public FileSelectView(Context context, int dialogid, CallbackBundle callback,
String suffix, Map<String, Integer> images) {
		super(context);
		this.imagemap = images;
		this.suffix = suffix==null?"":suffix.toLowerCase(Locale.getDefault());
		this.callback = callback;
		this.dialogid = dialogid;
		this.setOnItemClickListener(this);
		refreshFileList();
	}
	/**
	 * Query the suffix of file which want to filter
	 * @param filename
	 * @return
	 */
	private String getSuffix(String filename){
		int dix = filename.lastIndexOf('.');
		if(dix<0){
			return "";
		}
		else{
			return filename.substring(dix+1);
		}
	}

	/**
	 * Get The Image resource ID
	 * @param s
	 * @return
	 */
	private int getImageId(String s){
		if(imagemap == null){
			return 0;
		}
		else if(imagemap.containsKey(s)){
			return imagemap.get(s);
		}
		else if(imagemap.containsKey(sEmpty)){
			return imagemap.get(sEmpty);
		}
		else {
			return 0;
		}
```

```
        }
        /**
         * Refresh the file list in Window
         * @return
         */
        private int refreshFileList()
        {
                File[] files = null;
                try{
                        files = new File(path).listFiles();
                }
                catch(Exception e){
                        files = null;
                }
                if(files==null){
                        Toast.makeText(getContext(),
sOnErrorMsg,Toast.LENGTH_SHORT).show();
                        return -1;
                }
                if(list != null){
                        list.clear();
                }
                else{
                        list = new ArrayList<Map<String, Object>>(files.length);
                }

                ArrayList<Map<String, Object>> lfolders = new ArrayList<Map<String,
Object>>();
                ArrayList<Map<String, Object>> lfiles = new ArrayList<Map<String,
Object>>();

                if(!this.path.equals(sRoot)){
                        Map<String, Object> map = new HashMap<String, Object>();
                        map.put("name", sRoot);
                        map.put("path", sRoot);
                        map.put("img", getImageId(sRoot));
                        list.add(map);

                        map = new HashMap<String, Object>();
                        map.put("name", sParent);
                        map.put("path", path);
                        map.put("img", getImageId(sParent));
                        list.add(map);
                }

                for(File file: files)
                {
                        if(file.isDirectory() && file.listFiles()!=null){
                                Map<String, Object> map = new HashMap<String,
Object>();
                                map.put("name", file.getName());
                                map.put("path", file.getPath());
                                map.put("img", getImageId(sFolder));
                                lfolders.add(map);
                        }
                        else if(file.isFile()){
                                String sf =
getSuffix(file.getName()).toLowerCase(Locale.getDefault());
                                if(suffix == null || suffix.length()==0 || (sf.length()>0 &&
suffix.indexOf("."+sf+";")>=0)){
                                                Map<String, Object> map = new HashMap<String,
Object>();
                                                map.put("name", file.getName());
                                                map.put("path", file.getPath());
                                                map.put("img", getImageId(sf));
                                                lfiles.add(map);
                                }
                        }
                }

                list.addAll(lfolders);
                list.addAll(lfiles);


                SimpleAdapter adapter = new SimpleAdapter(getContext(), list,
R.layout.filedialogitem, new String[]{"img", "name", "path"}, new int[]{R.id.filedialogitem_img,
R.id.filedialogitem_name, R.id.filedialogitem_path});
                this.setAdapter(adapter);
                return files.length;
        }

        /**
         * OnItemClick action
         *
         * @see ListView#onItemClick(AdapterView<?> parent, View v, int position, long
id)
```

```
		 */
		@SuppressWarnings("deprecation")
		@Override
		public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
			String pt = (String) list.get(position).get("path");
			String fn = (String) list.get(position).get("name");
			if(fn.equals(sRoot) || fn.equals(sParent)){
				File fl = new File(pt);
				String ppt = fl.getParent();
				if(ppt != null){
					path = ppt;
				}
				else{
					path = sRoot;
				}
			}
			else{
				File fl = new File(pt);
				if(fl.isFile()){
					((Activity)getContext()).dismissDialog(this.dialogid);

					Bundle bundle = new Bundle();
					bundle.putString("path", pt);
					bundle.putString("name", fn);
					this.callback.callback(bundle);
					return;
				}
				else if(fl.isDirectory()){
					path = pt;
				}
			}
			this.refreshFileList();
		}
	}
}
```

**Persistence**

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.provider.BaseColumns;

import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttMessage;

import java.util.ArrayList;
import java.util.List;

/**
 * <code>Persistence</code> deals with interacting with the database to persist
 * {@link Connection} objects so created clients survive, the destruction of the
 * singleton {@link Connections} object.
 *
 */
public class Persistence extends SQLiteOpenHelper implements BaseColumns {

  /** The version of the database **/
  public static final int DATABASE_VERSION = 1;

  /** The name of the database file **/
  public static final String DATABASE_NAME = "connections.db";
  /** The name of the connections table **/
  public static final String TABLE_CONNECTIONS = "connections";

  /** Table column for host **/
  public static final String COLUMN_HOST = "host";
  /** Table column for client id **/
  public static final String COLUMN_client_ID = "clientID";
  /** Table column for port **/
  public static final String COLUMN_port = "port";
```

```
/** Table column for ssl enabled**/
public static final String COLUMN_ssl = "ssl";

//connection options
/** Table column for client's timeout**/
public static final String COLUMN_TIME_OUT = "timeout";
/** Table column for client's keepalive **/
public static final String COLUMN_KEEP_ALIVE = "keepalive";
/** Table column for the client's username**/
public static final String COLUMN_USER_NAME = "username";
/** Table column for the client's password**/
public static final String COLUMN_PASSWORD = "password";
/** Table column for clean session **/
public static final String COLUMN_CLEAN_SESSION = "cleanSession";
/** Table column for **/

//last will
/** Table column for last will topic **/
public static final String COLUMN_TOPIC = "topic";
/** Table column for the last will message payload **/
public static final String COLUMN_MESSAGE = "message";
/** Table column for the last will message qos **/
public static final String COLUMN_QOS = "qos";
/** Table column for the retained state of the message **/
public static final String COLUMN_RETAINED = "retained";

//sql lite data types
/** Text type for SQLite**/
private static final String TEXT_TYPE = " TEXT";
/** Int type for SQLite**/
private static final String INT_TYPE = " INTEGER";
/**Comma separator **/
private static final String COMMA_SEP = ",";

/** Create tables query **/
private static final String SQL_CREATE_ENTRIES =

   "CREATE TABLE " + TABLE_CONNECTIONS + " (" +
      _ID + " INTEGER PRIMARY KEY," +
      COLUMN_HOST + TEXT_TYPE + COMMA_SEP +
      COLUMN_client_ID + TEXT_TYPE + COMMA_SEP +
      COLUMN_port + INT_TYPE + COMMA_SEP +
      COLUMN_ssl + INT_TYPE + COMMA_SEP +
          COLUMN_TIME_OUT + INT_TYPE + COMMA_SEP +
          COLUMN_KEEP_ALIVE + INT_TYPE + COMMA_SEP +
          COLUMN_USER_NAME + TEXT_TYPE + COMMA_SEP +
          COLUMN_PASSWORD + TEXT_TYPE + COMMA_SEP +
          COLUMN_CLEAN_SESSION + INT_TYPE + COMMA_SEP +
          COLUMN_TOPIC + TEXT_TYPE + COMMA_SEP +
          COLUMN_MESSAGE + TEXT_TYPE + COMMA_SEP +
          COLUMN_QOS + INT_TYPE + COMMA_SEP +
          COLUMN_RETAINED + " INTEGER);";

  /** Delete tables entry **/
  private static final String SQL_DELETE_ENTRIES =
      "DROP TABLE IF EXISTS " + TABLE_CONNECTIONS;

  /**
   * Creates the persistence object passing it a context
   * @param context Context that the application is running in
   */
  public Persistence(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
  }

  /* (non-Javadoc)
   * @see
android.database.sqlite.SQLiteOpenHelper#onCreate(android.database.sqlite.SQLiteDatabase)
   */
  @Override
  public void onCreate(SQLiteDatabase db) {
    db.execSQL(SQL_CREATE_ENTRIES);

  }

  /* (non-Javadoc)
   * @see
android.database.sqlite.SQLiteOpenHelper#onUpgrade(android.database.sqlite.SQLiteDatabas
e, int, int)
   */
  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL(SQL_DELETE_ENTRIES);
  }

  /*
```

```
   * (non-Javadoc)
   * @see
android.database.sqlite.SQLiteOpenHelper#onDowngrade(android.database.sqlite.SQLiteDatab
ase, int, int)
   */
  @Override
  public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    onUpgrade(db, oldVersion, newVersion);
  }

  /**
   * Persist a Connection to the database
   * @param connection the connection to persist
   * @throws PersistenceException If storing the data fails
   */
  public void persistConnection(Connection connection) throws PersistenceException {

    MqttConnectOptions conOpts = connection.getConnectionOptions();
    MqttMessage lastWill = conOpts.getWillMessage();
    SQLiteDatabase db = getWritableDatabase();
    ContentValues values = new ContentValues();

    //put the column values object

    values.put(COLUMN_HOST, connection.getHostName());
    values.put(COLUMN_port, connection.getPort());
    values.put(COLUMN_client_ID, connection.getId());
    values.put(COLUMN_ssl, connection.isSSL());

    values.put(COLUMN_KEEP_ALIVE, conOpts.getKeepAliveInterval());
    values.put(COLUMN_TIME_OUT, conOpts.getConnectionTimeout());
    values.put(COLUMN_USER_NAME, conOpts.getUserName());
    values.put(COLUMN_TOPIC, conOpts.getWillDestination());

    //uses "condition ? trueValue: falseValue" for in line converting of values
    char[] password = conOpts.getPassword();
    values.put(COLUMN_CLEAN_SESSION, conOpts.isCleanSession() ? 1 : 0); //convert
boolean to int and then put in values
    values.put(COLUMN_PASSWORD, password != null ? String.valueOf(password) : null);
//convert char[] to String
    values.put(COLUMN_MESSAGE, lastWill != null ? new String(lastWill.getPayload()) : null); //
convert byte[] to string
    values.put(COLUMN_QOS, lastWill != null ? lastWill.getQos() : 0);
    if (lastWill == null) {
      values.put(COLUMN_RETAINED, 0);
    }
    else {
      values.put(COLUMN_RETAINED, lastWill.isRetained() ? 1 : 0); //convert from boolean to int
    }

    //insert the values into the tables, returns the ID for the row
    long newRowId = db.insert(TABLE_CONNECTIONS, null, values);

    db.close(); //close the db then deal with the result of the query

    if (newRowId == -1) {
      throw new PersistenceException("Failed to persist connection: " + connection.handle());
    }
    else { //Successfully persisted assigning persistecneID
      connection.assignPersistenceId(newRowId);
    }
  }

  /**
   * Recreates connection objects based upon information stored in the database
   * @param context Context for creating {@link Connection} objects
   * @return list of connections that have been restored
   * @throws PersistenceException if restoring connections fails, this is thrown
   */
  public List<Connection> restoreConnections(Context context) throws PersistenceException
  {
    //columns to return
    String[] connectionColumns = {
        COLUMN_HOST,
        COLUMN_port,
        COLUMN_client_ID,
        COLUMN_ssl,
        COLUMN_KEEP_ALIVE,
        COLUMN_CLEAN_SESSION,
        COLUMN_TIME_OUT,
        COLUMN_USER_NAME,
        COLUMN_PASSWORD,
        COLUMN_TOPIC,
        COLUMN_MESSAGE,
        COLUMN_RETAINED,
        COLUMN_QOS,
        _ID

    };

    //how to sort the data being returned
    String sort = COLUMN_HOST;

    SQLiteDatabase db = getReadableDatabase();

    Cursor c = db.query(TABLE_CONNECTIONS, connectionColumns, null, null, null, null, sort);
    ArrayList<Connection> list = new ArrayList<Connection>(c.getCount());
    Connection connection = null;
    for (int i = 0; i < c.getCount(); i++) {
      if (!c.moveToNext()) { //move to the next item throw persistence exception, if it fails
        throw new PersistenceException("Failed restoring connection - count: " + c.getCount() +
"loop iteration: " + i);
      }
      //get data from cursor
      Long id = c.getLong(c.getColumnIndexOrThrow(_ID));
      //basic client information
      String host = c.getString(c.getColumnIndexOrThrow(COLUMN_HOST));
      String clientID = c.getString(c.getColumnIndexOrThrow(COLUMN_client_ID));
      int port = c.getInt(c.getColumnIndexOrThrow(COLUMN_port));

      //connect options strings
      String username = c.getString(c.getColumnIndexOrThrow(COLUMN_USER_NAME));
      String password = c.getString(c.getColumnIndexOrThrow(COLUMN_PASSWORD));
      String topic = c.getString(c.getColumnIndexOrThrow(COLUMN_TOPIC));
      String message = c.getString(c.getColumnIndexOrThrow(COLUMN_MESSAGE));

      //connect options integers
      int qos = c.getInt(c.getColumnIndexOrThrow(COLUMN_QOS));
      int keepAlive = c.getInt(c.getColumnIndexOrThrow(COLUMN_KEEP_ALIVE));
      int timeout = c.getInt(c.getColumnIndexOrThrow(COLUMN_TIME_OUT));

      //get all values that need converting and convert integers to booleans in line using "condition
? trueValue : falseValue"
      boolean cleanSession = c.getInt(c.getColumnIndexOrThrow(COLUMN_CLEAN_SESSION))
== 1 ? true : false;
      boolean retained = c.getInt(c.getColumnIndexOrThrow(COLUMN_RETAINED)) == 1 ? true :
false;
      boolean ssl = c.getInt(c.getColumnIndexOrThrow(COLUMN_ssl)) == 1 ? true : false;
      //rebuild objects starting with the connect options
      MqttConnectOptions opts = new MqttConnectOptions();
      opts.setCleanSession(cleanSession);
      opts.setKeepAliveInterval(keepAlive);
      opts.setConnectionTimeout(timeout);

      opts.setPassword(password != null ? password.toCharArray() : null);
      opts.setUserName(username);

      if (topic != null) {
        opts.setWill(topic, message.getBytes(), qos, retained);
      }

      //now create the connection object
      connection = Connection.createConnection(clientID, host, port, context, ssl);
      connection.addConnectionOptions(opts);
      connection.assignPersistenceId(id);
      //store it in the list
      list.add(connection);

    }
    //close the cursor now we are finished with it
    c.close();
    db.close();
    return list;

  }

  /**
   * Deletes a connection from the database
   * @param connection The connection to delete from the database
   */
  public void deleteConnection(Connection connection) {
    SQLiteDatabase db = getWritableDatabase();

    db.delete(TABLE_CONNECTIONS, _ID + "=?", new
String[]{String.valueOf(connection.persistenceId())});
    db.close();
    //don't care if it failed, means it's not in the db therefore no need to delete

  }
}
```

### Persistence Exception

```
/*******************************************************************************
 * Copyright (c) 1999, 2014 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *    http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 */
package org.eclipse.paho.android.service.sample;

/**
 * Persistence Exception, defines an error with persisting a {@link Connection}
 * fails. Example operations are {@link Persistence#persistConnection(Connection)} and {@link
Persistence#restoreConnections(android.content.Context)};
 * these operations throw this exception to indicate unexpected results occurred when
performing actions on the database.
 *
 */
public class PersistenceException extends Exception {

  /**
   * Creates a persistence exception with the given error message
   * @param message The error message to display
   */
  public PersistenceException(String message) {
    super(message);
  }

  /** Serialisation ID**/
  private static final long serialVersionUID = 5326458803268855071L;

}
```

**LAYOUT FILES:**

### Activity_Advanced

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
-->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <LinearLayout
      android:id="@+id/usernameGroup"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <TextView
         android:id="@+id/textView1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/uname"
         android:layout_marginRight="15dip"/>

      <EditText
         android:id="@+id/uname"
                  android:layout_width="0dip"
         android:layout_height="wrap_content"
         android:ems="10"
         android:layout_weight="0.22"
         android:hint="@string/unameHint"
         android:inputType="text"
          >
      </EditText>
   </LinearLayout>

   <LinearLayout
      android:id="@+id/passwordGroup"
               android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_below="@id/usernameGroup"
    android:layout_marginTop="25dp" >

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pass"
        android:layout_marginRight="25dip" />

    <EditText
        android:id="@+id/password"
        android:layout_width="0dip"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_weight="0.22"
        android:hint="@string/passwordHint"
        android:inputType="textPassword" />
</LinearLayout>

<LinearLayout
    android:id="@+id/sslGroup"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/passwordGroup"
    android:layout_marginTop="25dp" >

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ssl"
        android:layout_marginRight="60dip" />

    <CheckBox
        android:id="@+id/sslCheckBox"
                android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/empty" />

    <EditText
        android:id="@+id/sslKeyLocaltion"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:enabled="false"
        android:hint="@string/sslKeyLocaltion"
        android:inputType="text" >

    </EditText>

    <Button
        android:id="@+id/sslKeyBut"
        style="?android:attr/buttonStyleSmall"
        android:clickable="false"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/select" />

</LinearLayout>

<LinearLayout
    android:id="@+id/timeoutGroup"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/sslGroup"
    android:layout_marginTop="25dp" >

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/timeout"
        android:layout_marginRight="35dip" />

    <EditText
        android:id="@+id/timeout"
                android:layout_width="0dip"
        android:layout_height="wrap_content"
        android:ems="10"
        android:layout_weight="0.22"
        android:hint="@string/timeoutHint"
        android:inputType="number" />
</LinearLayout>
```

```
    <LinearLayout
        android:id="@+id/keepaliveGroup"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/timeoutGroup"
        android:layout_marginTop="25dp" >

        <TextView
            android:id="@+id/textView5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/keepalive"
            android:singleLine="false"
            android:layout_marginRight="25dip"/>

        <EditText
            android:id="@+id/keepalive"
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:ems="10"
            android:layout_weight="0.22"
            android:hint="@string/keepaliveHint"
            android:inputType="number" />
    </LinearLayout>

</RelativeLayout>
```

**Activity_Connection_Details**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
<android.support.v4.view.ViewPager
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ConnectionDetails" />
```

**Activity_New_Connection**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
-->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText
        android:id="@+id/clientId"
        android:layout_width="250dip"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="text"
        android:layout_below="@+id/clientIDTextView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="81dp" />

    <TextView
        android:id="@+id/clientIDTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="10dip"
        android:text="@string/clientID"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="109dp" />

</RelativeLayout>
```

**Client_Connections**

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clickable="true"
    tools:context=".ClientConnections" >

</ListView>
```

**Connection_Text_View**

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:focusable="false"
    android:padding="10dp"
    android:drawableRight="@drawable/arrow"
/>
```

**File Dialog Item**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/vw1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#000000"
    android:orientation="horizontal"
    android:padding="4dp" >


    <ImageView
        android:id="@+id/filedialogitem_img"
        android:layout_width="32dp"
        android:layout_height="32dp"
        android:layout_margin="4dp"/>



   <LinearLayout
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:orientation="vertical" >





        <TextView
            android:id="@+id/filedialogitem_name"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textColor="#FFFFFF"
            android:textSize="18sp"
            android:textStyle="bold" />


        <TextView
            android:id="@+id/filedialogitem_path"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:paddingLeft="10dp"
            android:textColor="#FFFFFF"
            android:textSize="14sp" />

    </LinearLayout>

</LinearLayout>
```

**List_View_Text_View**

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="272dp"
    android:layout_height="fill_parent"
    android:focusable="false"
    android:padding="10dp"
/>
```

**Map_Frag**

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.gms.maps.MapView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/mapfrag"
    android:name="com.google.android.gms.maps.MapFragment"
    map:mapType="normal"
    />

<!--
```

```
xmlns:tools="http://schemas.android.com/tools"-->
   <!--    map:cameraTargetLat="41.7056"
      map:cameraTargetLng="-86.2353"
      map:cameraZoom="16"-->
```

**Notification**

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="#fea11e"
   >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceMedium"
      android:text="@string/RFIDname"
      android:id="@+id/RFID"
      android:layout_marginTop="105dp"
      android:layout_alignParentTop="true"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true"
      android:layout_marginLeft="42dp"
      android:layout_marginStart="42dp"
      android:textColor="#ffffff" />

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceMedium"
      android:text="@string/Itemname"
      android:id="@+id/Name"
      android:layout_marginTop="46dp"
      android:layout_below="@+id/RFID"
      android:layout_alignLeft="@+id/RFID"
      android:layout_alignStart="@+id/RFID"
      android:password="false"
      android:textColor="#ffffff" />


   <EditText
```

```
        android:layout_width="175dp"
        android:layout_height="wrap_content"
        android:id="@+id/NumberText"
        android:layout_alignBottom="@+id/RFID"
        android:layout_toEndOf="@+id/Name"
        />

    <EditText
        android:layout_width="175dp"
        android:layout_height="wrap_content"
        android:id="@+id/NameText"
        android:layout_alignBottom="@+id/Name"
        android:layout_toRightOf="@+id/Name"
        android:layout_toEndOf="@+id/Name"
        android:onClick="onClick" />
</RelativeLayout>
```

**MENU FILES:**

**Activity_Advanced**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
<menu xmlns:android="http://schemas.android.com/apk/res/android">

            <item
        android:id="@+id/ok"
        android:title="@string/save"
        android:showAsAction="always" />

                  <item
        android:id="@+id/setLastWill"
        android:title="@string/willMessage"
        android:titleCondensed="@string/willMessageShort"
        android:showAsAction="always" />
```

```
</menu>
```

**Activity_Client_Connections_Contextual**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
   <item android:id="@+id/delete"
        android:title="@string/delete"
       >

   </item>


</menu>
```

**Activity_Connection_Details**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
 <menu xmlns:android="http://schemas.android.com/apk/res/android">

<item android:id="@+id/disconnect" android:title="@string/Disconnect"
android:showAsAction="ifRoom"></item>
</menu>
```

**Activity_Connection_Details_Disconnected**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
```

```
IBM Corp.
 -->
 <menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/connectMenuOption" android:title="@string/connect"
android:showAsAction="ifRoom"></item>
</menu>
```

**Activity_Connections**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
 <menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/newConnection" android:enabled="true"
        android:titleCondensed="@string/newConnectionShort"
        android:title="@string/newConnection"
        android:showAsAction="always" >
    </item>
    <item android:id="@+id/startLogging" android:enabled="true"
        android:title="@string/startLogging"
        android:showAsAction="never">

    </item>
</menu>
```

**Activity_New_Connection**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
```

```
IBM Corp.
 -->
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/connectAction"
                                android:showAsAction="always"
         android:title="@string/connect" />
        <item
        android:id="@+id/advanced"
         android:title="@string/advanced"
        android:showAsAction="ifRoom"
        android:titleCondensed="@string/advancedShort"/>
</menu>
```

**Activity_publish**

```
<!--
Licensed Materials - Property of IBM

5747-SM3

(C) Copyright IBM Corp. 1999, 2012 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
 -->
<menu xmlns:android="http://schemas.android.com/apk/res/android">


        <item
        android:id="@+id/publish"
                    android:showAsAction="ifRoom"
        android:title="@string/publish" />
        <item android:id="@+id/disconnect" android:title="@string/Disconnect"
android:showAsAction="ifRoom"></item>
</menu>
```

**STRINGS FILE:**

**Strings**

```
<!--
Licensed Materials - Property of IBM
```

```
 -->
<resources>

    <string name="app_name">TrakPack</string>
    <string name="menu_settings">Settings</string>
    <string name="server">Server</string>
    <string name="port">Port</string>
    <string name="connect">Connect</string>

    <string name="newConnection">New Connection</string>
    <string name="newConnectionShort">+</string>

    <string name="title_activity_new_connection">New Connection</string>
    <string name="clientID">Please Enter Your Name</string>
    <string name="connectedto">Connected to </string>

    <string name="Disconnect">Disconnect</string>
    <string name="subscribe">Subscribe</string>
    <string name="publish">Rename</string>
    <string name="title_activity_subscribe">Subscribe</string>
    <string name="disconnected">Disconnected from</string>
    <string name="topic">Topic</string>
    <string name="history">History</string>
    <string name="title_activity_history">History</string>
    <string name="subscribed">Subscribed to </string>
    <string name="token">org.eclipse.paho.android.service.sample.ClientConnections</string>
    <string name="no_status">Unknown connection status to</string>
    <string name="connecting">Connecting to</string>
    <string name="disconnecting">Disconnecting from</string>
    <string name="connectionError">An error occurred connecting to</string>

    <string name="title_activity_publish">Publish</string>
    <string name="message">Message</string>
    <string name="qos0">0</string>
    <string name="qos1">1</string>
    <string name="qos2">2</string>
    <string name="empty"></string>
    <string name="qos">QOS</string>
    <string name="retained">Retained</string>
    <string name="cleanSession">Clean Session</string>
    <string name="advanced">Advanced Options</string>
    <string name="advancedShort">Advanced</string>
    <string name="title_activity_advanced">Advanced</string>
    <string name="uname">User Name</string>
    <string name="pass">Password</string>
    <string name="ssl">SSL</string>
    <string name="timeout">Time Out</string>
    <string name="keepalive">Keep Alive \nTimeout</string>

    <string name="title_activity_last_will">Last Will Message</string>
    <string name="setLastWill">Set Last Will Message</string>
    <string name="willMessage">Set Last Will Message</string>
    <string name="willMessageShort">Last Will</string>
    <string name="notifyTitle">Item Scanned</string>
    <string name="notifyTitle_connectionLost">MQTT Client Has Lost Connection</string>

    <string name="save">Save</string>
    <string name="title_activity_connection_details">Connection Details</string>
    <string name="startLogging">Enable Logging</string>
    <string name="endLogging">Disable Logging</string>
        <string name="missingOptions">ClientID, server address or Port number is missing.
Please correct or press back arrow cancel.</string>

    <string name='dateFormat'>dd/MM/yy \'at\' HH:mm:ss</string>
    <string name="unkown_error">Unknown Error</string>
    <string name="delete">Delete Connection</string>
    <string name="disconnectClient">Disconnect Client?</string>
    <string name="deleteDialog">The selected client is currently connected or connecting,
deleting this connection will cause it to be disconnected.</string>
    <string name="continueBtn">Continue</string>
    <string name="cancelBtn">Cancel</string>

    <!-- Strings that need formatting  -->
    <string name="messageRecieved">Your %1$s was scanned at %2$s. </string> <!--
Received message %1$s &lt;br/&gt; &lt;small&gt;Topic: %2$s &lt;/small&gt; -->
    <string name="timestamp">&lt;br/&gt; &lt;small&gt; %1$s &lt;/small&gt;</string>
    <string name="notification">%1$s scanned a %2$s at %3$s.</string>
    <string name="toast_pub_success">Published message: %1$s to topic: %2$s</string>
    <string name="toast_sub_success">Subscribed to %1$s</string>
    <string name="toast_disconnected">Disconnected</string>
    <string name="toast_pub_failed">Failed to publish message: %1$s to topic: %2$s</string>
    <string name="toast_sub_failed">Failed to subscribe to %1$s</string>
    <string name="connection_lost">%1$s has lost connection to %2$s</string>
    <string name="failure_disconnect">Disconnect failed.&lt;br/&gt; &lt;small&gt;Reason:
%s&lt;/small&gt; </string>
    <string name="failure_connect">Client failed to connect.&lt;br/&gt; &lt;small&gt;Reason:
%s&lt;/small&gt;</string>
    <string name="client_connected">Client connected successfully</string>

    <!-- Hints -->
    <string name="serverURIHint">example.example.com</string>
    <string name="portHint">1883</string>
    <string name="contentDescriptionSSL">Enable SSL</string>
    <string name="contentDescriptionCleanSession">Clean Session</string>
    <string name="topicHint">hello</string>
    <string name="messageHint">Hello World</string>
    <string name="unameHint">user</string>
    <string name="passwordHint">password</string>
    <string name="timeoutHint">60</string>
    <string name="keepaliveHint">200</string>
    <string name="sslKeyLocaltion">file://</string>
    <string name="select">select</string>
    <string name="title_activity_notification">Object Naming</string>
    <string name="title_maps_activity">Map</string>

    <!-- Trakpak Strings -->
    <string name="RFIDname">ID Number</string>
    <string name="Itemname">Object Name</string>
    <string name="google_app_id">AIzaSyDaFtfvSf2Kez4ycJgMkc03P0yO5o3vflo</string>
    <string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">AIzaSyDaFtfvSf2Kez4ycJgMkc03P0yO5o3vflo</string>

</resources>
```

## 8.3 Data Sheets

PIC Datasheet - This datasheet details the operation of the microcontroller.

Gtop GPS - These two datasheets contain the specifications used in our GPS unit.

https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf

Average Diode Forward Current - This was the information we used to account for our LED's.

ESP8266 - This is the datasheet for our WiFi component.

ID-12LA Datasheet - This is the datasheet for our RFID scanner.

5000 mAh LiPo - This is the datasheet for our power source.

MCP73831 - This is the datasheet for the recharging component.

TPS61201 - This is the datasheet for the DC-DC boost converter.