

# UNIVERSITY OF ALASKA ANCHORAGE

CSCE A470

CAPSTONE PROJECT

## Arduino GPS Logging Midterm Report

Author:

**Kenneth Mendenhall**

Supervisor:

**Prof Adriano Cavalcanti, PhD**

Anchorage AK, April 2016



Computer Science &  
Engineering Department  
UNIVERSITY *of* ALASKA ANCHORAGE

© Copyright 2016

by

Kenneth Mendenhall

[kcmendenhallii@alaska.edu](mailto:kcmendenhallii@alaska.edu)

## **Abstract**

In recent years the rise in power of microcontrollers, both in speed and memory capacity, has served to drive the “Do-It-Yourself” (DIY) electronics movement, sparking a generation of amateur hobbyists and inventors the means to make household projects to better their lives and further their knowledge of digital systems. Further, this rise in demand for higher performance micro systems along with the shields, peripheral devices, to accompany them has also spurred the evolution of devices like the Raspberry Pi and Intel Edison into systems that can compete with low end desktop systems, but at a much cheaper cost. Additionally, the DIY surge can serve as an educational tool to enlighten future generations of potential coders and computer systems engineers at an earlier age, and has prompted the formation of several home-town maker organizations to facilitate awareness and education in the digital age.

## **Acknowledgements**

I would like to show my utmost gratitude for my mom, Leslie Mendenhall, and my dad, Ken Mendenhall, who both encouraged me all throughout my formative years to learn and study as much as possible and who were critical of my work early on. I would also like to extend my gratitude to the Computer Science faculty and staff who gave me the tools necessary to make this project a reality.

Firstly, I would like to acknowledge my supervisor for the Capstone project, Professor Adriano Cavalcanti, PhD. whose patience and mentorship guided the development of my research project. Secondly, I would like to acknowledge Prof. Frank Moore, PhD. who instructed my first computer science courses and who laid the foundation for the rest of my education at the University of Alaska Anchorage.

## Contents

<b>Cover Page</b> .....	1
<b>Abstract</b> .....	3
<b>Acknowledgements</b> .....	4
<b>Table of Contents</b> .....	5
<b>Table List</b> .....	6
<b>Figure List</b> .....	7
<b>Chapter 1</b> .....	8
<b>Chapter 2</b> .....	12
<b>Chapter 3</b> .....	18
<b>Chapter 4</b> .....	21
<b>Chapter 5</b> .....	24
<b>Appendix A</b> .....	27
<b>Appendix B</b> .....	28
<b>References</b> .....	39

## **Table List:**

Table 1.1 Describes the flow of data through the program as a function of inputs for the service live of each object throughout the life of the programs execution .....	11
Table 2.1 Shows the projected timeline for task completion, modified to meet the existing demands of a compressed schedule .....	17

## Figure List

Figure 1.1 shows the Intel Edison with development breakout board.....	9
Figure 1.2 shows the Xbee module that will facilitate the data link between the Java application and the remote Edison device .....	9
Figure 1.3 Shows the GP-17340 unit that will be utilized, as sized relative to a quarter.....	10
Figure 1.5 Is an image of the GPS network that has been created to facilitate navigation.....	10
Figure 2.1 Is the logo for the software environment used to configure the XBee radios. Developed by Digi, International.....	12
Figure 2.2 Shows some of the configurations that can be specified in the firmware for the XBee Pro inside the XCTU interface.....	12
Figure 2.3 Is the iconic symbol for the concept of the "Internet-Of-Things" .....	13
Figure 2.4 Examples GPS Data being fed from a receiver using standard NMEA Strings.....	14
Figure 3.1 Shows a preliminary design for the GUI using a simplistic button set for the interface.....	18
Figure 3.2 Shows a minimal segment of .kml code, sufficient enough to drop a placemark on a map, as prescribed by Google API.....	19
Figure 5.1 Shows two XBee Radios and a 9 Axis-IMU, all future expansions to the GPS Logging program.....	25

# Chapter 1

## Introduction

---

### 1.1 Introduction

The implementation of a program to record the latitude and longitude positions from a GPS device by using open source tools (NMEA Parsing Library, Xbee Interface, Intel Edison Toolkit, etc) is a relatively inexpensive project for such broad reaching application. The utilization of a microprocessor with breakout board allows end users and DIY hobbyists to expand on the given system to meet the needs of whatever project they are working on. This enables users to receive real time positioning, course, speed, and even altitude data and record this information. The motivation for using the Intel Edison system and making a program totally from scratch as opposed to using existing API's for Apple or Android stems from the desire to leave the project open-ended for future expansion.





*Figure 1.1 shows the Intel Edison with development breakout board*

## 1.2 Application

The Arduino GPS Logging project is composed of a Java application running on a PC and interfacing a Intel Edison processor equipped with a GPS-13740, linked via 2 Xbee radios, to allow the device to be dislocated from the user if need be. The Java application establishes a serial communication with the PC side Xbee using open source API supplied by Digi, while also handling the writing of incoming data to a .kml file. The Edison side of the project has a corresponding Xbee that has been configured to the same network as the other, and is used to receive commands and transmit position data back to the user. The Edison supplies position data using open source API supplied by the National Marine Electronics Association, to parse incoming GPS strings and extracting pectinate data, packaging it into messages and then sending it off to the Xbee for transmission.



*Figure 1.2 Shows the Xbee Radio module that will facilitate the data link between the Java application and the remote Edison device*

## 1.3 Motivation

The intent of this project is to assist DIY/amateur hobbyists by providing an open source set of instructions on how configure a GPS tracking program for any desired application. As previously

stated, the relatively low budget required to get this project off the ground is sufficient for most newcomers who want to experiment with hardware, as opposed to being limited by software.

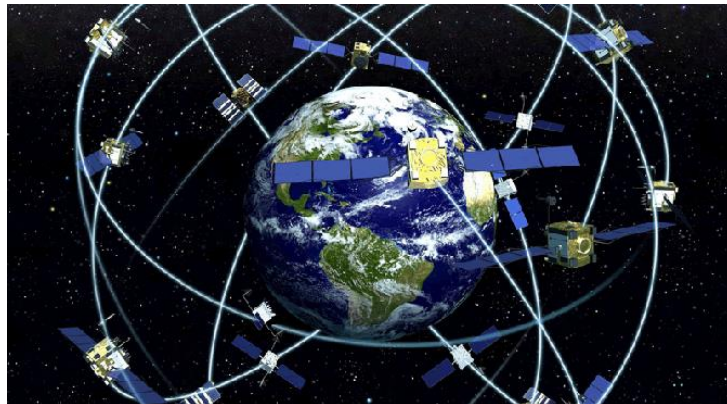


*Figure 1.3 Shows the GP-17340 unit that will be utilized, as sized relative to a quarter*

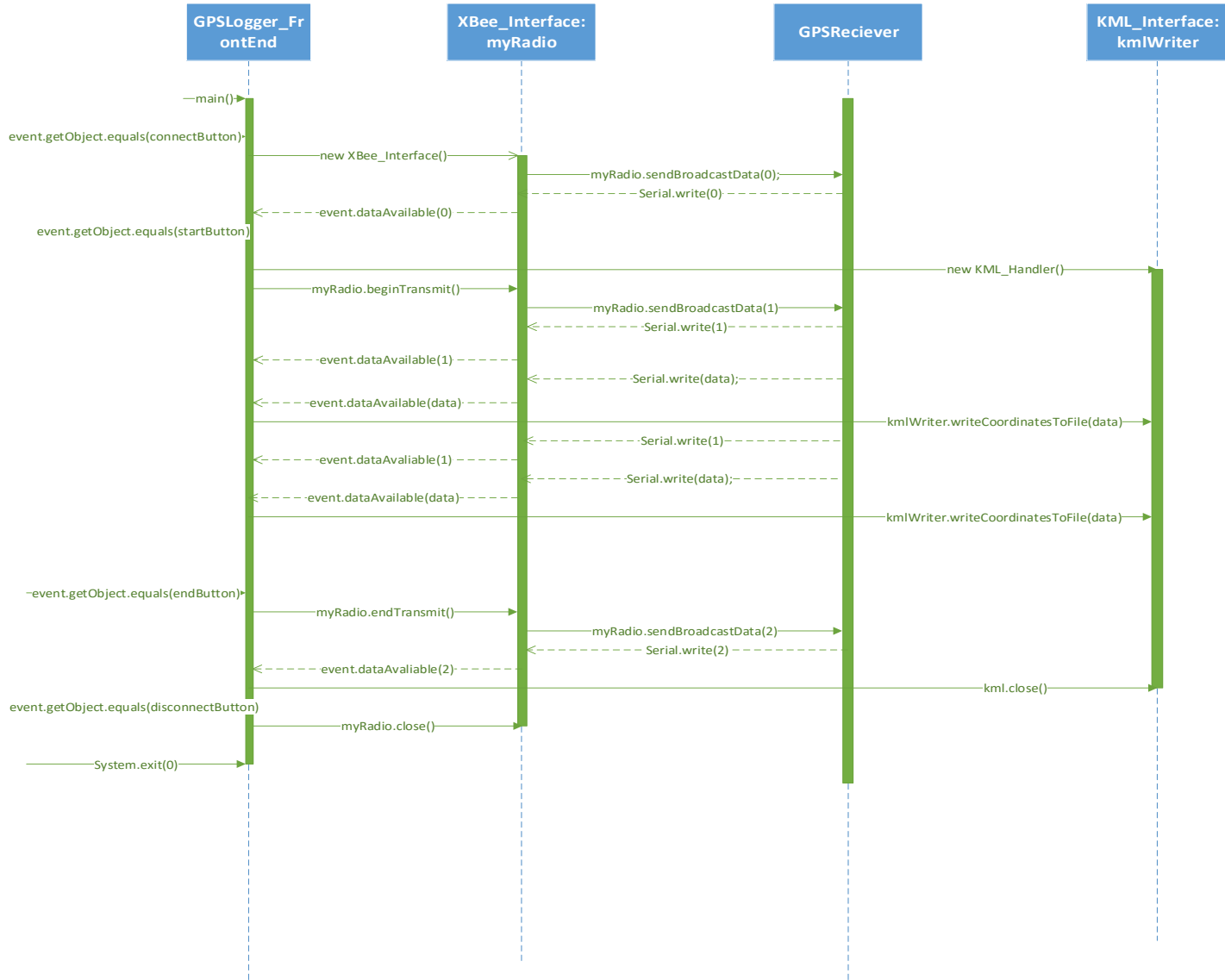
Further, Edison can be configured to use the Arduino IDE which implements a form of the C programming language and is a great tool to teach others who are new to programming. This simple design philosophy comes full circle with the Arduino GPS Logging program which Additionally, working across multiple systems presents a new challenge, along with familiarizing myself with Keyhole Markup Language, which is used by Google Earth to log Geographic data. With Intel Edison breakout board the possibilities for future projects is endless.

## **1.4 Recent Developments**

Right now the Java side code is being integrated with the Edison side C code for passing of GPS data. There is a Bitbucket account for hosting developments on the project as well as source code



*Figure 1.4 Is an image of the GPS network that has been created to facilitate navigation*



*Table 1.1 Describes the flow of data through the program as a function of inputs for the service live of each object throughout the life of the programs execution*

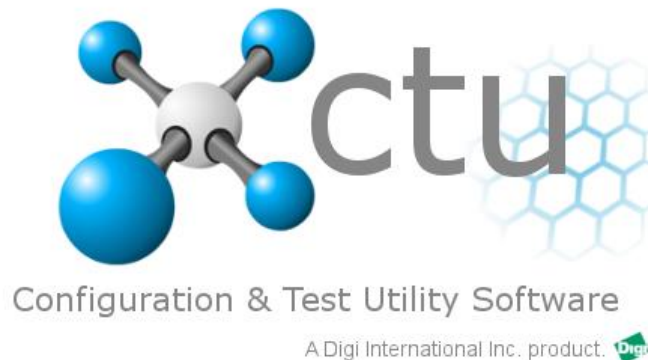
## Chapter 2

# Implementation and Design

---

### 2.1 Introduction

Integration of the systems in the Arduino GPS Logging program will be aided by various open source drivers and libraries. Such libraries will facilitate wireless data exchange, serial communication, parsing and writing data to .kml files, as well as reading GPS data. All of the provided data was acquired from various websites on the internet.



*Figure 2.1 Is the logo for the software environment used to configure the XBee radios.  
Developed by Digi, International*

The Java program itself has been compiled and tested on Netbeans 8.0.2, while the Arduino code was written on the Arduino IDE 1.6.6 and deployed on the Intel Edison board running the FlashEdison JSON Image.



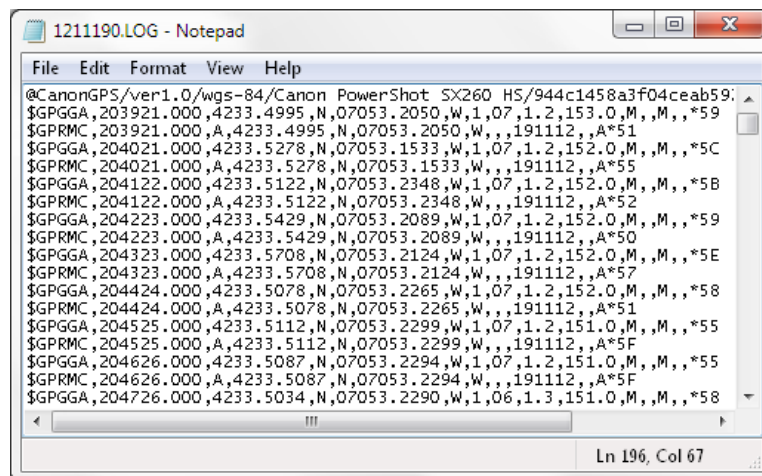
## 2.2 Application

Once the required libraries are imported to Netbeans the application is ready to be tested. This involves making a simple GUI for the front end with a button only interface for user interaction. The buttons for interaction thus far are:

- Connect: performs a sequence of actions to initialize an Xbee object and connect to the Edison end system.
- Start: initiates a command to the Edison to tell the device that the host is ready for receipt of GPS data.
- End: initiates a command to the Edison to tell the device that the host is no longer needing GPS data.
- Disconnect: Terminates the connection with the Xbee radios and saves the supplied data to the .kml file and closes the file connection.

This can be simplified by further abstraction in future applications by automatically performing the connect and disconnect functions, requiring the user to only initiate the point at which the program needs to start/stop logging coordinates.

Once the connection has been established the user is ready to request data from the GPS. These are supplied by simple byte of data that tells the Edison when the host is ready for receipt.



The image shows a Notepad window titled "1211190.LOG - Notepad". The window contains a list of NMEA strings, which are standard GPS data formats. The strings include information such as time, latitude, longitude, speed, and altitude. The window's status bar at the bottom indicates "Ln 196, Col 67".

```
@CanonGPS/ver1.0/wgs-84/Canon PowerShot SX260 HS/944c1458a3f04ceab59:
$GPGGA,203921.000,4233.4995,N,07053.2050,W,1,07,1.2,153.0,M,,*59
$GPRMC,203921.000,A,4233.4995,N,07053.2050,W,,191112,,A*51
$GPGGA,204021.000,4233.5278,N,07053.1533,W,1,07,1.2,152.0,M,,*5C
$GPRMC,204021.000,A,4233.5278,N,07053.1533,W,,191112,,A*55
$GPGGA,204122.000,4233.5122,N,07053.2348,W,1,07,1.2,152.0,M,,*58
$GPRMC,204122.000,A,4233.5122,N,07053.2348,W,,191112,,A*52
$GPGGA,204223.000,4233.5429,N,07053.2089,W,1,07,1.2,152.0,M,,*59
$GPRMC,204223.000,A,4233.5429,N,07053.2089,W,,191112,,A*50
$GPGGA,204323.000,4233.5708,N,07053.2124,W,1,07,1.2,152.0,M,,*5E
$GPRMC,204323.000,A,4233.5708,N,07053.2124,W,,191112,,A*57
$GPGGA,204424.000,4233.5078,N,07053.2265,W,1,07,1.2,152.0,M,,*58
$GPRMC,204424.000,A,4233.5078,N,07053.2265,W,,191112,,A*51
$GPGGA,204525.000,4233.5112,N,07053.2299,W,1,07,1.2,151.0,M,,*55
$GPRMC,204525.000,A,4233.5112,N,07053.2299,W,,191112,,A*5F
$GPGGA,204626.000,4233.5087,N,07053.2294,W,1,07,1.2,151.0,M,,*55
$GPRMC,204626.000,A,4233.5087,N,07053.2294,W,,191112,,A*5F
$GPGGA,204726.000,4233.5034,N,07053.2290,W,1,06,1.3,151.0,M,,*58
```

Figure 2.4 Examples GPS Data being fed from a receiver using standard NMEA Strings

Example string:

```
$GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
```

The data being provided by the GP-13740 receiver will be in a \$GPRMC format (One of several supplied formats specified by NMEA). These strings include:

- \$GPRMC is the header for the type of data to follow
- Time of the GPS fix (UTC).
- Receiver error byte in the form of the characters A (Success) or V (Receiver Failure).
- Latitude (N/S) and Longitude (E/W) data, in degree decimal format.
- Computed speed over the ground in various units (for this project m/s will be used).
- Degrees will be the computed course relative to True North. Following the course data will be the
- Date of the fix
- Magnetic variation data (E/W).
- Mandatory checksum bit ensures data integrity.

GPS strings are parsed using the NMEA supplied methods:

- `gps.gprmc_latitude();`
- `gps.gprmc_longitude();`
- `gps.gprmc_course();`
- `gps.gprmc_speed(MPS);`

\*Note: The parameter MPS specifies Meters Per Second for computation by the NMEA library\*

Once parsed, the strings are sent back to the host via the Xbee radios and encoded into the .kml files for users to glance at later via Google Earth,

## 2.3 Motivation

As previously stated the intent of this project is to explore the use of microcontrollers and integrating them with software projects and sharing a common passion for DIY development of electronic systems over the internet for others to learn from.

## **2.4 Recent Developments**

Right now the Java side code is being integrated with the Edison side C code for passing of GPS data. There is a Bitbucket account for housing developments on the project as well as source code.

## **2.5 Agile as a Coding Methodology**

Agile describes a means to go about planning and executing coding projects. These projects best implemented through Agile if they are smaller in scope and can be flexible in their requirements. Agile necessitates smaller functional teams of coders, working together, and taking on distinct roles within the group. This method to coding is also effective when team leadership can regularly meet with the clients, who are needing the final product. Typically, the clients and team leadership will initially meet and negotiate on the scale and scope of the product, as well as establish timelines and requirements for each iteration, until completion.

From a coding perspective, the Agile methodology is efficient because it does not demand up from costs, such as verbose documentation, that may have to be changed if the scope of the project changes. Further, Agile fosters a sleek, efficient approach to coding by making teams adaptable to hardships, i.e. teammates dropping out mid stride, and by promoting workload management for the team through burndown charts.





# Chapter 3

## Interface Design Methodology

### 3.1 Introduction

Development of the user interface is inspired by simplicity of design and functionality for the end user. As described in Chapter 2, the graphic user interface for the Arduino GPS Logging program consists of four buttons: connect, start, end, disconnect. These provide the key functions to the back end sub systems when actuated.

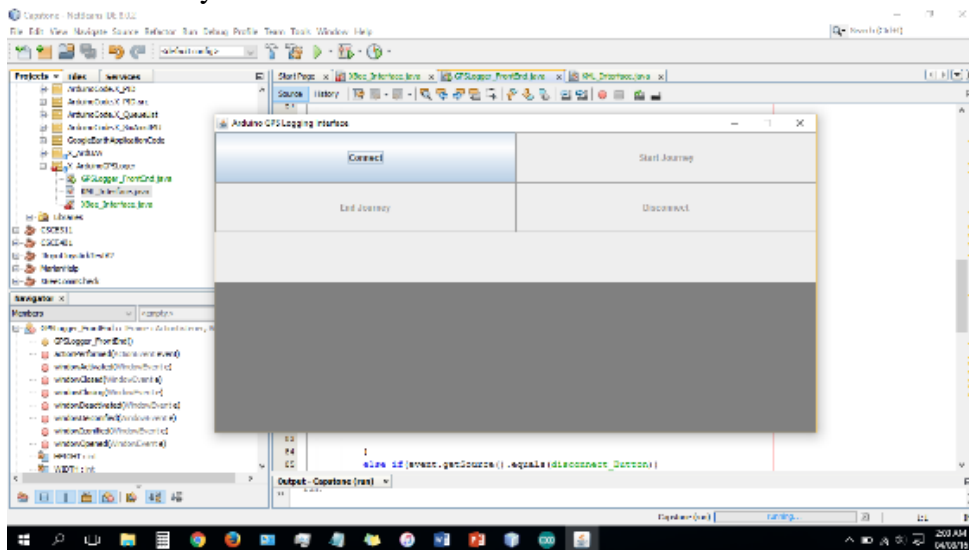


Figure 3.1 Shows a preliminary design for the GUI using a simplistic button set for the interface

## 3.2 Application

At the start of the program the only button enabled is the connect button. This prevents users from actuating other controls out of sequence and makes testing easier and implementation more predictable. Once the connect button is clicked, an event is fired off on the back end and caught inside the GPSLogger\_FrontEnd (the GUI). The method associated with the connect button initializes the Xbee radio as an object and initiates a handshake to the Intel Edison system. This turns on the GPS and allows it to warm up (could take several seconds). Once a signal is acquired a reply is sent back to the FrontEnd and the Start button becomes enabled. Again, this makes testing easier and prevents the user from taking inappropriate or illogical actions.

With the start button enabled, the user can then command the Edison to start transmitting GPS data to the FrontEnd. When the user clicks the start button the program first initializes the KML\_Interface object as a means of handling the GPS data when it starts coming in. The KML\_Interface prescribes a means of taking GPS data in inserting the Latitude/Longitude coordinates into pre-defined segments of the Keyhole Markup Language text. These segments are then written to a file that can then be read in Google Earth to communicate geographic data and visually represent data points in space.

```
111 |
112 |         file.writeBytes("<Placemark>\n"
113 |                         + "<name>" + trackCount + "</name>\n"
114 |                         + "<open>1</open>\n"
115 |                         + "<styleUrl>#msn_track</styleUrl>\n"
116 |                         + "<Point>\n"
117 |                         + "<gx:drawOrder>1</gx:drawOrder>\n"
118 |                         + "<coordinates>" + longIn + "," + latIn + ",0</coordinates>\n"
119 |                         + "</Point>\n"
120 |                         + "</Placemark>\n"
121 |                         + "</Document>\n"
122 |                         + "</kml>");
123 |         trackCount++;
124 |     }
125 |     catch (IOException e)
126 |     {
127 |         System.out.println("Error amending " + pathLogger.getPath() + pathLogger.getName());
128 |     }
129 | }
```

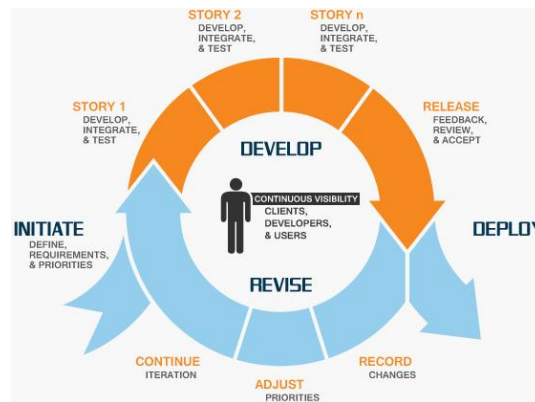
Figure 3.2 Shows a minimal segment of .kml code, sufficient enough to drop a place mark on a map, as prescribed by Google API

Upon clicking the start button, the end button becomes enabled, and the start button becomes disabled. The Arduino program will continue to transmit GPS coordinates until the end button is pressed. At this point the end button will activate an event and will cause the Xbee\_Interface to send a byte sequence to the Edison to stop transmitting. This will be acknowledged by the Edison controller so that the host will know to finish writing to the .kml file. Pressing the end

button will also enable the start button again so that the user will have the option to continue receiving GPS data. The user also can terminate the program by pressing the disconnect button. The design of the program is such that there is no need for direct human interaction with the Arduino system, and that the supplied tools (GUI) are sufficient to meet the internet of a capable interface.

### 3.3 Agile Methodology

Given that this project is the offspring of another project I was working on, a lot of last minute adjustments had to be made to cut back on the size and scope, leaving just this boiled down GUI as the result. Given that that decision was also made so late in the game (End of February), the team had to be flexible and realistic enough in changing its course to allow time to complete all the required documentation (validating another point for Agile, in not holding the team bound to volumes of pseudocode at the onset of development).



*Table 3.3 Shows the general flow of the Agile coding lifecycle throughout project development*

However, given the rapid change in direction of the project, the team had not had sufficient time to inform Dr. Cavalcanti of its intent when he signed on to be the project supervisor. Given this constraint, the best is being made do with the current situation.

### 3.4 Motivation

Again, as previously stated, the intent of this project is to develop a base set of tools that can be integrated later into future projects. And, working with microcontrollers can be a great learning aid for individuals who are just starting out in the DIY community.

### 3.5 Recent Developments

Testing of the Edison system will begin soon. The link for the Bitbucket account hosting source code will be found at: [https://Codemonkey40k@bitbucket.org/csce407\\_capstonegroup/master.git](https://Codemonkey40k@bitbucket.org/csce407_capstonegroup/master.git)

## Chapter 4

# Arduino GPS Program User Manual and Instillation Guide

---

### 4.1 Introduction

This chapter is a supplement to setting up the Arduino GPS Logging Program, configuring all devices and equipment, and loading the applicable libraries. The intent of this section is to inform the user about configuration data to get the project up and running smoothly. While the development team for this project recognizes that the instructions require much on the end user do themselves, it may be a future goal for this project to wrap all start up utilities into one zip and introduce an auto-install feature to make set-up as seamless as possible.

### 4.2 Application

The Arduino GPS Logging system was created by Kenneth Mendenhall for CSCE 407 Capstone course at the University of Alaska Anchorage. There is no differentiation in the versions yet as all source is under one Master Branch on the hosted BitBucket webpage.

### 4.3 Setup

This section is a walkthrough of the required components for successful operation of the Arduino GPS Logging program, as well as providing URL links to resources to acquire those

components. This is to save the user (potentially) hours of head scratching and page turning across the internet to find resources and instructions for each part, just to get started.

- The classes here-in make use of Java RXTXComm.jar found at [http://rxtx.qbang.org/wiki/index.php/Main\\_Page](http://rxtx.qbang.org/wiki/index.php/Main_Page)

- Place the RXTXComm.jar on local machine here:

C:\Program Files\Java\jdk1.8.0\_66\jre\lib\ext

- Additional files required will be rxtxSerial.dll. The 64-bit version called 'RXTX native driver' can be found here:

<http://jlog.org/rxtx-win.html>

- And placed on local machine here:

C:\Program Files\Java\jdk1.8.0\_66\jre\bin

- Download the Arduino IDE, or latest version thereof, can be obtained from the Arduino home page at:

<https://www.arduino.cc/en/Main/Software>

- Download of the Intel Edison® Installer and image was acquired from, along with the installation tutorial:

<https://software.intel.com/en-us/get-started-edison-windows-step2>

- Transcribe the libraries over to the (default on setup of Arduino installation) directory:

C:\Program Files (x86)\Arduino\libraries

- XBee radios will need to be configured using the XCTU interface and can be acquired from Digi's webpage at:

<http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>

- To put devices on the same network each device MAC address will need to be configured inside the XCTU interface so that the devices know who to look for.

-

- While using Netbeans (or the IDE of preference) add the libraries to your project and import the code.

## **4.3 Motivation**

The motivation for this section is that the required components must be configured for the Arduino / Intel Edison® to be able to operate correctly.

Additional objectives of this section will be to find the error recording rate of the GP-17340 receiver as well as seeing if additional accuracy can be obtained by eliminating such error causing situations. This will be done by finding situations, via frequent usage, where the GPS receiver transmits faulty / inaccurate data, finding common elements across those situations, and then finding ways to mitigate faulty readings for future applications. This was not an immediate objective on the project, but will be necessary for future implementations of the project and its further development and application.

## **4.4 Recent Developments**

Right now the Java side code is being integrated with the Edison side C code for passing of GPS data. There is a Bitbucket account for hosting developments on the project as well as source code.

## Chapter 5

# Arduino GPS Logging Program Summary and Conclusions

---

### 5.1 Introduction

The Arduino GPS Project was devised as an attempt to make a consolidated resource for interfacing Google Earth with the data coming from a GPS unit, in order to realize path information in a simple manner. In order to do this an understanding of how geographic data is represented in Google Earth was acquired, by studying the Keyhole Markup API, and a means of making modifications to such Keyhole Markup files was constructed.

In addition to the afore mentioned functionality, the Arduino GPS Logging system had to interface a user with an Arduino based platform. The project accounted for this platform being dislocated from the user, however, successful implementation of the remote control functionality was not made possible, and local testing was done via a hardline USB cable. Future development will modify this flaw.

### 5.2 Lessons Learned

The Arduino GPS Project was initially planned as part of a larger development that involved using a two-way communications channel to interface the computer side program with the Arduino platform. As such, this bigger project would have accounted for the user plotting a path in Google Earth and saving the path in a .kml file. The program would have been able to extract the path, as an array of latitude / longitude points, and send it to the Arduino unit.



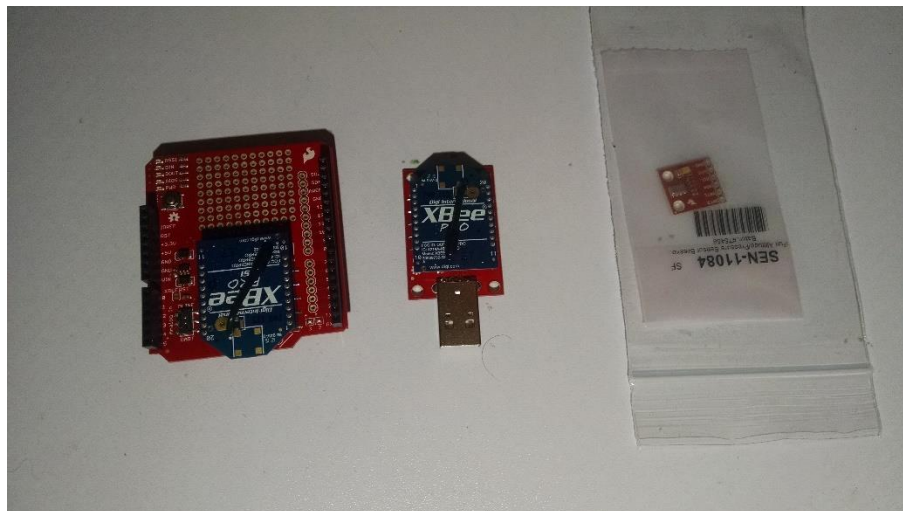
Future development could include a self navigating system that receives a set of waypoints and, after acquiring a GPS fix, could navigate along this assigned path using the GPS data as a point of reference.

A lot of problems were encountered in development of this project, some were overcome, others not. The GPS module became unable to receive a valid fix, therefore, in the middle of development modifications had to be made to the code, so that it would still execute regardless of the lack of signal. Future work with the Arduino GPS Logging Program will possibly require the acquisition of a new GPS module to replace the current one.

Other problems encountered were that the XBee radio module could not be used in the project at the same time with the GPS module. This is due to the fact that both of the libraries designed to support their hardware made use of the Rx port (Pin 0) on the Edison Breakout Development Board. Future development on this end will necessitate the possible use of the SoftwareSerial library, supplied as default for the Arduino, to read from a separate PWM pin and forward the data on to the data parsing program for the GPS.

### 5.3 Implications

As previously stated the Arduino GPS Data Logging program was intended for a larger project. Work will be conducted over the next several months to incorporate the Data Logger into a functional, autonomous, self navigating system. Such work, hopefully, will have broader applications in the field of Search and Rescue or Wildlife Management in Alaska. It is the hope of the project developers that this system will be put to good use.

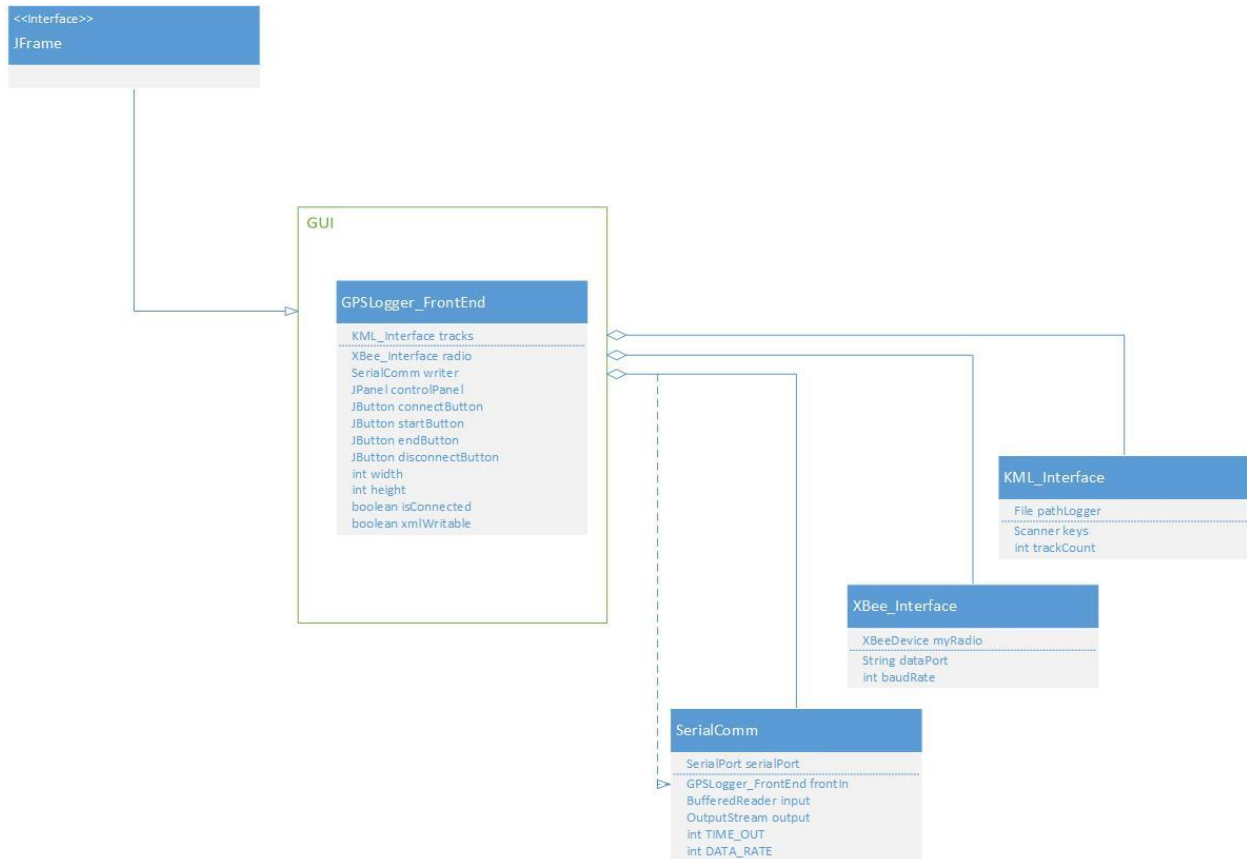


*Figure 5.1 Shows two XBee Radios and a 9 Axis-IMU, all future expansions to the GPS Logging program*

Based on feedback I got from the poster session, one feature that will need further work is the analysis of power consumption with the GPS and Edison systems, and how to separate them

from the application user without the USB teather. Figure 5.1 presents a visual of future modules to expand the Data Logging program with.

# Appendix A



*Appendix A: Shows the high level structure of the ArduinoGPS\_Logging program and all of its dependencies*

## Appendix B

```
package X_ArduinoGPSLogger;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import com.digi.xbee.api.XBeeDevice;
import com.digi.xbee.api.exceptions.XBeeException;

import gnu.io.CommPortIdentifier;
import gnu.io.SerialPort;
import gnu.io.SerialPortEvent;
import gnu.io.SerialPortEventListener;

public class GPSLogger_FrontEnd extends JFrame implements ActionListener, WindowListener
{
    private KML_Interface tracks;
    private XBee_Interface radio;
    private SerialComm writer;

    private final int WIDTH = 850, HEIGHT = 450;
    private final JPanel gpsControlPanel;
    private final JButton connect_Button, start_Button, end_Button, disconnect_Button;
    private boolean isConnected = false, xmlWritable = false;

    public GPSLogger_FrontEnd()
    {
        Container arduContentPanel = getContentPane();
        writer = new SerialComm(this);
        tracks = new KML_Interface(this);

        while(writer.initialize() == false)
        {
            System.out.println("Unable to find Port. Exiting now.");
            System.exit(0);
        }

        arduContentPanel.setLayout(new GridLayout(2,2));
    }
}
```

```

arduContentPanel.setBackground(Color.gray);

setTitle("Arduino GPS Logging Interface");
setSize(WIDTH, HEIGHT);
setLocation(300,250);
setResizable(false);
isDisplayable();
toFront();
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

addWindowListener(this);

//Button Panel////////////////////////////////////
gpsControlPanel = new JPanel(new GridLayout(3,1));

connect_Button = new JButton("Connect");
connect_Button.addActionListener(this);

start_Button = new JButton("Start");
start_Button.addActionListener(this);
start_Button.setEnabled(false);

end_Button = new JButton("End");
end_Button.addActionListener(this);
end_Button.setEnabled(false);

disconnect_Button = new JButton("Disconnect");
disconnect_Button.addActionListener(this);
disconnect_Button.setEnabled(false);

gpsControlPanel.add(connect_Button);
gpsControlPanel.add(start_Button);
gpsControlPanel.add(end_Button);
gpsControlPanel.add(disconnect_Button);

gpsControlPanel.setVisible(true);

add(gpsControlPanel);
setVisible(true);

}

public void actionPerformed(ActionEvent event)
{
    byte temp;

```

```

if(event.getSource().equals(connect_Button))
{
    //radio = new XBee_Interface();
    temp = 48;
    writer.serialWrite(temp);

    start_Button.setEnabled(true);
    connect_Button.setEnabled(false);

    isConnected = true;
}
else if(event.getSource().equals(start_Button))
{
    temp= 49;
    writer.serialWrite(temp);

    end_Button.setEnabled(true);
    start_Button.setEnabled(false);
    disconnect_Button.setEnabled(false);

}
else if(event.getSource().equals(end_Button))
{
    temp = 50;
    writer.serialWrite(temp);

    end_Button.setEnabled(false);
    start_Button.setEnabled(true);
    disconnect_Button.setEnabled(true);
}
else if(event.getSource().equals(disconnect_Button))
{
    temp = 51;
    writer.serialWrite(temp);

    isConnected = false;
    System.exit(0);
}
}

public void windowOpened(WindowEvent e)
{
}

```

```

public void windowActivated(WindowEvent e)
{
}
public void windowIconified(WindowEvent e)
{
}
public void windowDeiconified(WindowEvent e)
{
}
public void windowDeactivated(WindowEvent e)
{
}
public void windowClosed(WindowEvent e)
{
    if(isConnected = true)
    {

        System.out.println("System is currently in use. Premature termination is not
authorized.\n"
            + "Please click \"End\" then \"Disconnect\".");
    }
}
public void windowClosing(WindowEvent e)
{
}
}

class GPS_Interface
{
    public static void main(String[] args)
    {
        GPSLogger_FrontEnd logger = new GPSLogger_FrontEnd();
    }
}

```

```

package X_ArduinoGPSLogger;

import java.io.IOException;
import java.util.Scanner;
import org.jsoup.Jsoup;
import org.jsoup.parser.Parser;
import org.jsoup.nodes.Document;
import org.jsoup.select.Elements;
import java.util.ArrayList;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.RandomAccessFile;
import java.io.File;
//import javax.swing.JFileChooser;
//import javax.swing.filechooser.FileSystemView;
import javax.swing.*;

/**
 * Name: Kenneth Mendenhall
 * Date: 27NOV15
 * Course: CSCE 470 Capstone Project
 * University of Alaska Anchorage
 *
 * Purpose:
 * This program is designed to augment a package of tools for the ArdUAV
 * package.
 *
 * This class provides functionality to read and write to separate files
 * for the purposes of:
 * - transcribing received coordinates as a means of visually tracking flight
 * in a Google Earth application
 * - Reading coordinates from a file for the purpose of loading a flight
 * path to the ArdUAV drone
 */
public class KML_Interface
{
    private int trackCount = 0;
    private final Scanner keys;
    private final GPSLogger_FrontEnd frontEnd;
    private final File pathLogger;

    KML_Interface(GPSLogger_FrontEnd frontIn)
    {
        keys = new Scanner(System.in);
        frontEnd = frontIn;
    }

```



```

String filePath = "C:\\Users\\Kenneth\\Desktop\\UAA\\Capstone";
File dir = new File(filePath);

JFileChooser waypointFile = new JFileChooser();
waypointFile.setCurrentDirectory(dir);

JOptionPane.showMessageDialog(frontEnd, "This is a one time selection. Please select
a .kml file\n"
                                + "so the program can log the devices path.");
waypointFile.showDialog(frontEnd, "Select file to log path");

if(waypointFile.getSelectedFile() != null)
{
    pathLogger = waypointFile.getSelectedFile();
}
else
{
    while(waypointFile.getSelectedFile() == null)
    {
        waypointFile = new JFileChooser();
        waypointFile.setCurrentDirectory(dir);
        JOptionPane.showMessageDialog(frontEnd, "You must select a file to configure the
program!");
        waypointFile.showDialog(frontEnd, "Select file to log UAV flight path");
    }
    pathLogger = waypointFile.getSelectedFile();
}
}

//create functionality to write to/modify existing file
//for the purposes of logging waypoints from travel
//Generic .kml code for Waypoint storage looks like:
/*
<Placemark>
    <name>Track[i]</name>
    <open>1</open>
    <styleUrl>#msn_track</styleUrl>
    <Point>
        <gx:drawOrder>1</gx:drawOrder>
        <coordinates>-149.7444444444444,61.20777777777779,0</coordinates>
    </Point>
</Placemark>

*/
void writeCoordinatesToFile(double latIn, double longIn)

```

```

{
  try
  {
    RandomAccessFile file = new RandomAccessFile(pathLogger, "rw");

    //Constant offset 'file.length()-21' from the end of file
    //to place the write position of the file writer just before:
    // </Document>
    // </kml>"
    //so that new coordinates can be loaded to the file without overwriting
    //or potentially corrupting existing data
    file.seek(file.length()-19);

    file.writeBytes("<Placemark>\n"
      + "<name>" + trackCount + "</name>\n"
      + "<open>1</open>\n"
      + "<styleUrl>#msn_track</styleUrl>\n"
      + "<Point>\n"
      + "<gx:drawOrder>1</gx:drawOrder>\n"
      + "<coordinates>" + longIn + "," + latIn + ",0</coordinates>\n"
      + "</Point>\n"
      + "</Placemark>\n"
      + "</Document>\n"
      + "</kml>");
    trackCount++;
  }
  catch(IOException e)
  {
    System.out.println("Error ammending " + pathLogger.getPath() +
pathLogger.getName());
  }
}

void writeCoordinatesToFile(ArrayList<Double> coords)
{
  try
  {
    RandomAccessFile file = new RandomAccessFile(pathLogger, "rw");

    for(int i = 0; i < coords.size(); i+=2)
    {
      //Constant offset 'file.length()-21' from the end of file
      //to place the write position of the file writer just before:
      // </Document>
      // </kml>"
      //so that new coordinates can be loaded to the file without overwriting

```

```

//or potentially corrupting existing data
file.seek(file.length()-19);

file.writeBytes("<Placemark>\n"
    + "<name>" + trackCount + "</name>\n"
    + "<open>1</open>\n"
    + "<styleUrl>#msn_track</styleUrl>\n"
    + "<Point>\n"
    + "<gx:drawOrder>1</gx:drawOrder>\n"
    + "<coordinates>" + coords.get(i) + "," + coords.get(i+1) +
",0</coordinates>\n"
    + "</Point>\n"
    + "</Placemark>\n"
    + "</Document>\n"
    + "</kml>");
    trackCount++;
}
}
catch(IOException e)
{
    System.out.println("Error ammending " + pathLogger.getPath() +
pathLogger.getName());
}
}

```

```

//Return null if error
//otherwise returns list of lat/long coordinates
// works but for insert into correct position
// Ammendmet from original version:
// This function will now parse coordinates from a
// "path" from Google Earth rather than having
// individual waypoints.

```

```

boolean parseCoordinatesFromFile()
{
    //Ideally get file filter working to scrub out the other garbage

    //FileFilter filter = new FileNameExtensionFilter(null, "kml");
    String filePath = "C:\\Users\\Kenneth\\Desktop\\UAA\\Capstone";
    File dir = new File(filePath);

    JFileChooser waypointFile = new JFileChooser();
    waypointFile.setCurrentDirectory(dir);
    //waypointFile.setFileFilter(filter);
    int val = waypointFile.showDialog(frontEnd, "Select");
}

```

```

if(val == JFileChooser.APPROVE_OPTION)
{
    try
    {
        StringBuilder buf = new StringBuilder();

        try (BufferedReader in = new BufferedReader(new
FileReader(waypointFile.getSelectedFile())))
        {
            String str;

            while ((str = in.readLine()) != null)
            {
                buf.append(str);
            }
            in.close();
        }

        String html = buf.toString();

        Document doc = Jsoup.parse(html, "", Parser.xmlParser());
        Elements list = doc.select("coordinates");
        ArrayList<Double> coords = new ArrayList<>();

        String temp = list.toString().replace("<coordinates>", "").replace("</coordinates>",
");
        String[] tempArray = temp.split(",");

        tempArray[0] = tempArray[0].replaceAll("\\s+", "");
        coords.add(Double.parseDouble(tempArray[0]));
        coords.add(Double.parseDouble(tempArray[1]));

        for(int i = 2; i < tempArray.length-1; i+=2)
        {
            tempArray[i] = tempArray[i].substring(1, tempArray[i].length()-1);
            coords.add(Double.parseDouble(tempArray[i]));
            coords.add(Double.parseDouble(tempArray[i+1]));
        }

        //for testing purposes
        writeCoordinatesToFile(coords);

        //figure out way to handle comm flow between drone and base
        //via CommSuite class

```

```

        return true;
    }
    catch (IOException | NumberFormatException e)
    {
        JOptionPane.showMessageDialog(frontEnd, "Error was caught in
parseCoordinatesFromFile()\n" +
                                "and in parsing flight path from .kml file.");
        return false;
    }
}
else
{
    JOptionPane.showMessageDialog(frontEnd, "You did not select a waypoint file. You
may continue to fly the drone\n "
                                + "manually but will need a loaded set of waypoints to enable
Auto-Pilot");

    return false;
}
}

/*
public static void main(String[] args)
{
    XMLHandler parser = new XMLHandler(null);
    Scanner keys = new Scanner(System.in);

    System.out.print("Please enter a valid filepath: ");

    try
    {
        double[] coords = parser.parseCoordinatesFromFile(keys.nextLine());

        if(coords == null)
            return;

        for(int i = 0; i < coords.length; i+=3)
        {
            System.out.println("Latitude is: " + coords[i]);
            System.out.println("Longitude is: " + coords[i+1]);
        }
    }
    catch(Exception e)
    {
        System.out.println("An error occurred in accessing file");
    }
}

```

```
    parser.writeCoordinatesToFile(12, 12);  
  }  
  */  
}
```

## References

Electronics, My Funny (2014). *How to Configure XBee Modules For Simple Peer-to-Peer Communication*. Date accessed: April, 2016. <https://www.youtube.com/watch?v=77LTEjP-S6Y>

Intel (2016). *Assembling the Intel Edison® Board With the Arduino\* Expansion Board*. Date Accessed: April, 2016. <https://software.intel.com/en-us/assembling-intel-edison-board-with-arduino-expansion-board>

Olaussen, Arve (2011). *RXTX for Windows*. Date Accessed: April, 2016. <http://jlog.org/rxtx-win.html>

International, Digi (2016). *XCTU: Next Generation Configuration Platform for Xbee/RF Solutions*. Date Accessed: April, 2016. <http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>

Arduino (2016), *Arduino UNO & Genuino UNO*. Date Accessed: April, 2016. <https://www.arduino.cc/en/main/arduinoBoardUno>