

Digital Waste Management System
Brandon Lowe, Brian Yim, Kyle Lee, Lu Gan
University of British Columbia
EECE 409/429/419/439/400/469
July 15, 2014

Disclaimer: "UBC SEEDS provides students with the opportunity to share the findings of their studies, as well as their opinions, conclusions and recommendations with the UBC community. The reader should bear in mind that this is a student project/report and is not an official document of UBC. Furthermore readers should bear in mind that these reports may not reflect the current status of activities at UBC. We urge you to contact the research persons mentioned in a report or the SEEDS Coordinator about the current status of the subject matter of a project/report".

Digital Waste Management System

Brandon Lowe, Brian Yim, Kyle Lee, Lu Gan
Team 47



The Alma Mater Society (AMS) of the University of British Columbia (UBC) is building a new Student Union Building (SUB). As a means of measuring solid waste production at the SUB, a digital waste management system consisting of a floor scale, button panel and computer software backend is designed, allowing the weight, waste stream, date and time to be recorded and saved on a server. The system is also functionally tested at all stages of development to verify the correctness of the solution. The digital waste management system aims to be cheaper and faster than the alternative of conducting periodic waste audits, all the while being safe and easy to use.

Table of Contents

| | |
|---|----|
| List of Illustrations | 3 |
| List of Abbreviations | 4 |
| Glossary | 5 |
| 1.0 Introduction | 6 |
| 2.0 Scope and Purpose | 7 |
| 3.0 Architectural Design | 8 |
| 3.1 Floor Scale and Indicator | 9 |
| 3.2 Button Panel | 10 |
| 3.3 Emulated Server | 12 |
| 3.4 Computer Application | 13 |
| 3.5 Dashboard Simulator | 15 |
| 4.0 Formal Testing | 16 |
| 5.0 Risk Management | 17 |
| 6.0 Client Hand-Off | 18 |
| 7.0 Conclusion | 19 |
| Appendix A. Drawings and Schematics | |
| Appendix B. Source Code | |
| Appendix C. Testing Documents and Other Documentation | |
| Appendix D. User Manual | |

List of Illustrations

| | |
|--|----|
| Figure 1. High Level Block Diagram | 8 |
| Figure 2. Button Panel Inside View | 10 |
| Figure 3. Server Table Example | 12 |
| Figure 4. Computer Application Settings Window | 14 |
| Figure 5. Dashboard Simulator Graphs | 15 |
| Figure 6. Dashboard Simulator Graphical User Interface | 15 |

List of Abbreviations

AMS: Alma Mater Society

UBC: University of British Columbia

SUB: Student Union Building

ECE: Electrical and Computer Engineering

PCB: Printed circuit board

I²C: Inter-Integrated Circuit

IT: Information technology

GUI: Graphical user interface

Glossary

| | |
|--------------------|--|
| Alma Mater Society | A body at the University of British Columbia in charge of operating student services, as well as representing student needs and issues. |
| Waste stream | The different categories of waste, as a means of classifying the waste produced. In this project, four solid waste streams will be considered: Recyclables, paper, organics and other. |
| Waste audit | A process where all of the waste produced by a building is collected for a period of time, and then the waste is sorted and measured. A formal report is then produced, showing the different weights and statistics regarding all of the waste streams. |
| Indicator | An electronic device which powers a floor scale and accounts for calibration and units. The Cardinal 204 Indicator that is used in this project also supports communication with a computer to send weight data over a serial port connection. |
| Tare | A constant amount of weight to subtract from the gross measurement of a load, in order to account for the weight of an empty bin. The net weight, or the weight of the contents, is the gross weight minus the tare. |

1.0 Introduction

The Alma Mater Society* (AMS) is building a more sustainable Student Union Building (SUB). To promote sustainability, a means of measuring the waste produced by the four solid waste streams* (recyclables, paper, organics and other) is needed. However, the traditional method of conducting waste audits to measure waste production is both time-consuming and expensive, so an alternative is needed. As such, a digital waste management system will be created, aiming to be faster and cheaper than the waste audit*. The system will also store weight data on a server, where it can be displayed to the public via an electronic dashboard.

The required measurements are a weight of up to 500 kilograms, the stream name and the date and time of the measurement. This data must be stored locally and on a server provided by the AMS. The users will consist of janitorial staff, and as such the system must be easy to use. There are approximately 100 waste bins to be measured, so the measurements must be taken in under 10 seconds to maximize efficiency. The system must also be robust and have an expected lifetime of 10-15 years. The project duration is nine months, and the budget allocated for this project is \$650 from the Electrical and Computer Engineering (ECE) department and \$5000 from the AMS. The goals focus on the ability to port the system to other buildings.

The design consists of a waste scale and indicator*, a button panel and a computer to support the back end software to interface between the components. A private server will be emulated in order to provide the functionality of the AMS server until it is procured. The waste scale, indicator and button panel will be stored on-site in the waste facility, and the computer will be stored in a separate room for safety and security. In addition, a dashboard simulator will be designed, in order to show that data on the server can be pulled and interpreted.

* This and other terms marked with an asterisk will be defined in the Glossary on page 5

2.0 Scope and Purpose

The purpose of the project is to design a method of measuring the weight of the four solid waste streams produced by the SUB so that the data can ultimately be shown to the public via graphs and statistics to promote sustainability. The required measurements are the weight (up to 500kg), stream, date and time of each waste bin recorded, and these measurements must be able to be taken in under 10 seconds. This data must be stored on a server hosted by the AMS, as well as backed up locally. The system is operated by janitorial staff, so we must minimize the amount of physical labour involved. In addition, it must be easy to learn and operate to maximize the user buy-in. Because the system will be placed in the waste facility, it must also be safe and unobtrusive, as well as durable enough to have an expected lifetime of 10-15 years.

The time frame for the project is nine months. A total of \$5000 has been allocated to this project from the AMS, and \$650 has been allocated by the ECE department. Since the AMS is providing the server, we will have to ensure that our system is compatible with it. In addition, the electronic dashboard has not yet been designed and is being developed by a third party firm, so we will have to ensure that our data is kept in a sufficiently organized manner so that it conforms to the specifications of the dashboard. Because of the status of these third-party systems, our project will first be constructed as a stand-alone system, with a private emulated server to store data on.

The most important goal is extend our stand-alone system by designing a dashboard simulator to show that data on our server can be pulled and interpreted. Also of importance is ensuring that the system can be ported to other buildings and is adaptable. We should allow for streams to be changed, added and removed, as also take into consideration the waste bin tares*.

3.0 Architectural Design

The waste management system consists of a floor scale and indicator, a button panel and backend software installed on a computer. Janitorial staff will pull a waste bin across the scale and use the button panel to measure the weight of the bin, and then pull the bin off the scale and to the waste collection area after a successful measurement. Until the AMS procures the server in which the waste stream, weight, time and date will be stored, an emulated server is used to provide the same functionality. In addition, a dashboard simulator is designed to show that data can be pulled and manipulated from the server. The high level design for the implementation of the project is shown in Figure 1.

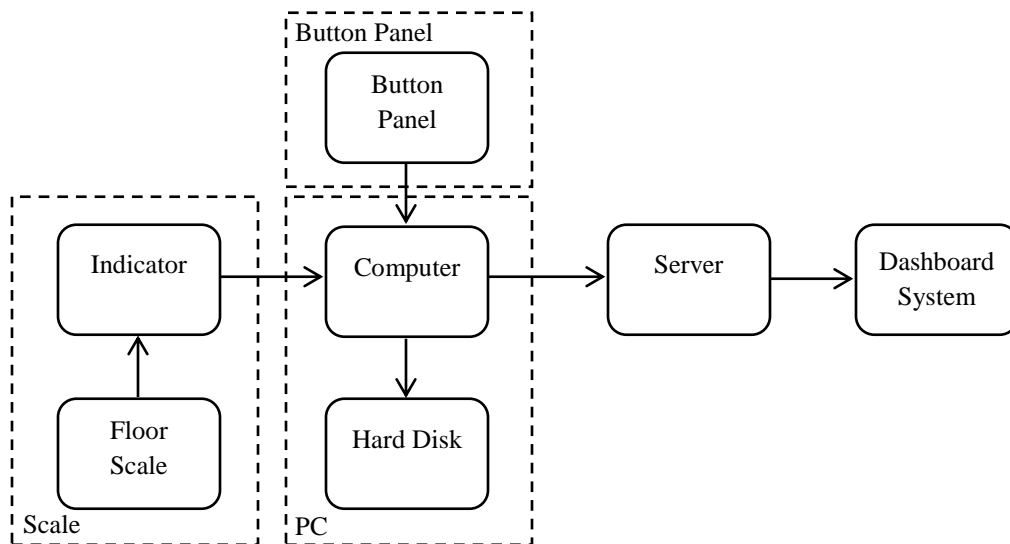


Figure 1. High level design of the digital waste management system

The final implementation of the system will instead use a server hosted by the AMS, and will also include an electronic dashboard instead of the dashboard simulator. The dashboard will function similar to the dashboard simulator we have designed, although it will include more statistics and graphics.

3.1 Floor Scale and Indicator

In order to measure the weight of the waste bins, a floor scale is needed. The scale must be large enough to safely support a waste bin without a high risk of the bin falling off the scale, and the load cells in the scale must have a capacity of at least 500 kilograms.

An Anyload FSP 36" x 36" floor scale is used, as it is large enough to support a bin and has a capacity of 500 kilograms. The scale is made of powder-coated steel, so it is sufficiently water-resistant to operate in the waste facility. The floor scale is top-loading, meaning that the scale will not undergo as much lateral stress as a scale with built-in ramps would, and as a result the expected lifetime is much longer. In addition, the scale is inexpensive compared to other scales on the market and is locally available.

A floor scale typically requires an indicator in order to power the scale, process the output signal from the scale's load cells and account for calibration. A Cardinal 204 Indicator is used, as it has a lot of documentation and is capable of outputting weight data on-demand through an RS-232 serial connection. The indicator connects to the floor scale via a 9-pin D connection and is powered by a DC adapter. The indicator is set up so that a string containing the weight data along with any status flags that denote information such as whether the weight was stable or unstable is sent through RS-232 upon receiving a weight request (0x05 in hexadecimal) from the computer connected to it. The string can then be parsed by the computer.

A top-loading scale requires ramps to be attached so that the user can approach the scale without having to lift the load. Two ramps are created out of aluminum checker plate, and these ramps sit beside the scale. Aluminum is used because it is more lightweight and easier to work with than steel, and as such the ramps are more portable. The ramps also contain guard rails to lower the risk of a waste bin falling off the side of the ramp.

3.2 Button Panel

A button panel is mounted on a wall beside the waste scale, as a means of allowing for the user to select the waste stream and to provide visual feedback regarding the status of the system. The button panel must be durable enough to operate in the waste facility, and must be able to send and receive data from a computer, so that the user input and waste data can all be processed.

A 10" x 10" x 4" electrical box is used to contain the electrical components, push buttons and Arduino microcontroller. Five 22mm plastic push buttons from Automation Direct are used for the waste stream selection, as they come in enough colours to be able to colour-code the different waste streams. Another 22mm push button is used as the confirmation button. These buttons, as well as a 7-segment display and a printed circuit board (PCB) are mounted on brackets that are attached to the electrical box. Bi-colour LEDs are used to display the currently selected waste stream, and single colour LEDs are used to display the status of the system. The layout of the electrical box is shown in Figure 2. In addition to the parts attached to the box, a 2.1mm barrel jack connector and female RS-232 connector are mounted on the side of the box.

The lid of the box is water-jet cut to contain holes for the push buttons, 7-segment display, LED indicator lights and labels.

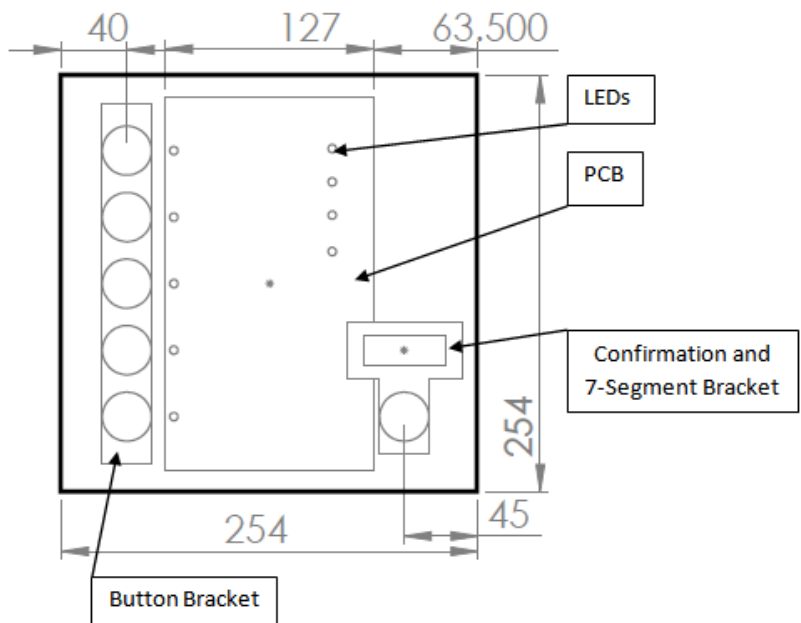


Figure 2. An overhead view of the inside of the button panel

The button panel is controlled by an Arduino Uno microcontroller. The Arduino contains a state machine which reads and debounces the push buttons, and also controls the LEDs and 7-segment display. Since the Arduino Uno has a limited number of I/O pins, four of the bi-colour stream LEDs are operated through a 74HCT4051 8-input analog demultiplexer connected to pins 2-5 on the Arduino. The 7-segment display is controlled through the Inter-Integrated Circuit (I²C) bus on analog pins 4-5, using the Adafruit LED Backpack libraries. The Arduino communicates with the computer using an RS-232 shield for the Arduino. Handshaking is achieved by the button panel waiting for a response from the computer application before registering a button press or state change. For example, if the user presses the Stream 1 button, the button panel will notify the computer, and only when the button panel receives a confirmation message from the computer will it reflect the state change, ensuring that both the button panel and computer keep the same state.

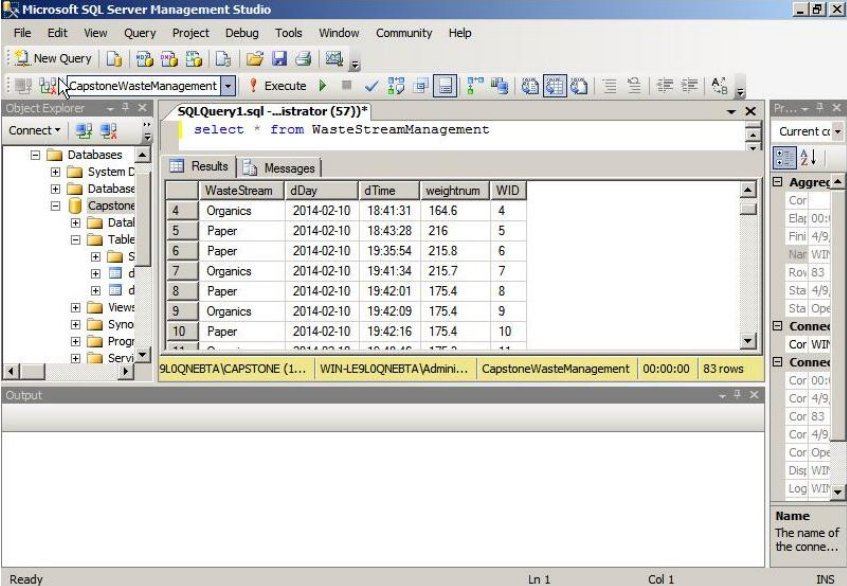
The 5 stream push buttons are connected to a 10 pin header (refer to Component P6 in the PCB Schematic in Appendix A) and the confirm button is connected to a 2 pin header. The five stream push button headers on the PCB are connected with 10k pull-up resistors to analog pins 0 to 3 of and digital pins 5 to 6 on the Arduino. Each of the 9 LEDs are separately mounted directly into their own respective 2 pin headers. The stream LED headers are connected to a 16 pin DIP socket for the demultiplexer. The demultiplexer is then wired to digital pins 2 to 5 on the Arduino. The status LED headers are connected with 220 ohm limiting resistors to digital pins 8 to 11 on the Arduino. Analog pins 4 and 5 on the Arduino is connected to four pin header (refer to Component P7 in the PCB Schematic in Appendix A) to drive the I²C which runs the 7-seg displays.

3.3 Emulated Server

The main goal of the digital waste management system is to collect data to display to the public. The data will be displayed on an electronic dashboard developed by a third-party firm, but to do so a server is needed to store the waste data. Since the AMS server is not yet ready, an emulated server is set up according to specifications provided by the client. The server is emulated through Virtual Box off-site and uses Windows Server 2008 R2 64 bit as the operating system and Microsoft SQL server 2008 as the backend to run the database.

An application is designed and coded in Java utilizing the JDBC library to set up and maintain a connection to the server through either local area network or through the internet. A list of server functions is created in a separate Java class and is responsible for sending queries to the server from which data could then be entered, deleted or retrieved.

The database consists of a table to store values from the user inputs. The columns of the table include fields for the stream name, date, time, weight and a unique identification number (WID). The stream, date, time and weight are provided by the computer application, and the unique identification number is automatically generated upon inserting a table entry. The table allows the computer to input data into it, query for specific data sorted by stream, time and date and deletion of certain inputs. Figure 3 shows a sample table.



The screenshot shows the Microsoft SQL Server Management Studio interface. The 'Results' pane displays the following data:

| | WasteStream | dDay | dTime | weightnum | WID |
|----|-------------|------------|----------|-----------|-----|
| 4 | Organics | 2014-02-10 | 18:41:31 | 164.6 | 4 |
| 5 | Paper | 2014-02-10 | 18:43:28 | 216 | 5 |
| 6 | Paper | 2014-02-10 | 19:35:54 | 215.8 | 6 |
| 7 | Organics | 2014-02-10 | 19:41:34 | 215.7 | 7 |
| 8 | Paper | 2014-02-10 | 19:42:01 | 175.4 | 8 |
| 9 | Organics | 2014-02-10 | 19:42:09 | 175.4 | 9 |
| 10 | Paper | 2014-02-10 | 19:42:16 | 175.4 | 10 |

Figure 3. Example SQL table for storing waste samples

3.4 Computer Application

A desktop computer is used to act as the middleman for all of the components of the system; the waste scale and indicator, the button panel and the server. An application is created to interface between these components, compatible with 64 bit Windows 7. The users should not have to interact with the computer other than to start up the software, as the button panel handles all of the external user input needed to weigh the waste bins. AMS Information Technology (IT) will be responsible for operating the software.

An application is designed using Java to interface between the waste scale and indicator, and to write data to the server once a successful weight measurement has been taken. The application consists of RS-232 serial port drivers for the indicator and button panel, a driver to access the server as well as a graphical user interface (GUI).

The RXTX serial port libraries are used to implement the drivers for the indicator and button panel. A superclass is created to implement the basic RS-232 connection and communication functions, and both the indicator and button panel drivers extend this superclass.

The button panel driver is responsible for handshaking with the button panel and handling the user input received from the button panel. Event-based serial communication is used in order to save computer processor time, and upon receiving data from the button panel, the data is parsed and the appropriate action is taken, either changing the currently selected waste stream or confirming and saving the weight sample. The button panel driver also interfaces with the indicator driver, and is responsible for updating the currently selected stream and tare for the indicator driver.

The indicator driver sends a weight request to the indicator every 500 milliseconds, and then parses the weight data into a weight and various status flags. The parsed weight is sent to

the button panel through the button panel driver, so that the current weight can be displayed to the user. The indicator driver also handles the status flags that the indicator sends alongside the weight sample, and as such is responsible for handling the scale errors such as the weight being negative or over capacity. In addition, once the user presses the confirmation button, the indicator will asynchronously send a weight request to ensure that the most recent weight sample is saved. If the weight sample is valid and the user has selected a stream, the weight, stream, time and date are then saved in a text file on local memory, as well as written to the server.

A GUI is developed to allow the IT to adjust the application settings as well as view a log of events and errors. The settings are broken down into different sections, as shown in Figure 4. In particular, the waste stream settings allow the user to enable or disable streams, as well as edit the names and tare values. In the event that a new waste bin model is introduced into the SUB and there are two different types of waste bins, an option exists for a second tare value to be implemented. All of the settings can be saved in a file called “settings.txt” and these settings are automatically loaded upon application start-up. In addition, the application can be started in either online or offline mode; offline mode will disable any server functionalities.

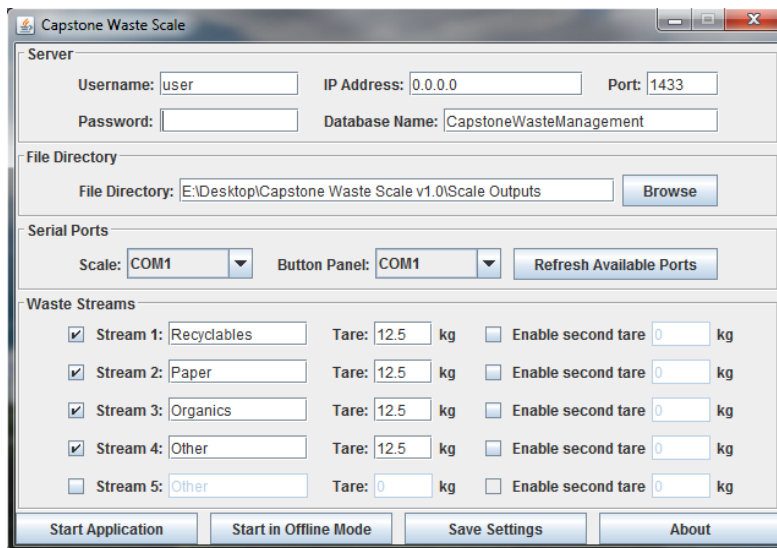


Figure 4. The settings screen for the computer application

3.5 Dashboard Simulator

A simple dashboard program that displays data collected in the form of graphs (shown in Figure 5) is created for demonstrative purposes. The dashboard program is written in Java and runs as an executable .jar file. The application connects to the SQL server and sends queries for each of the 4 streams between the dates specified by the user. The data is manipulated and stored into a single variable, then displayed using the JFreeChart libraries. A simple GUI is created for the user to specify dates and type of graph, as shown in Figure 6.

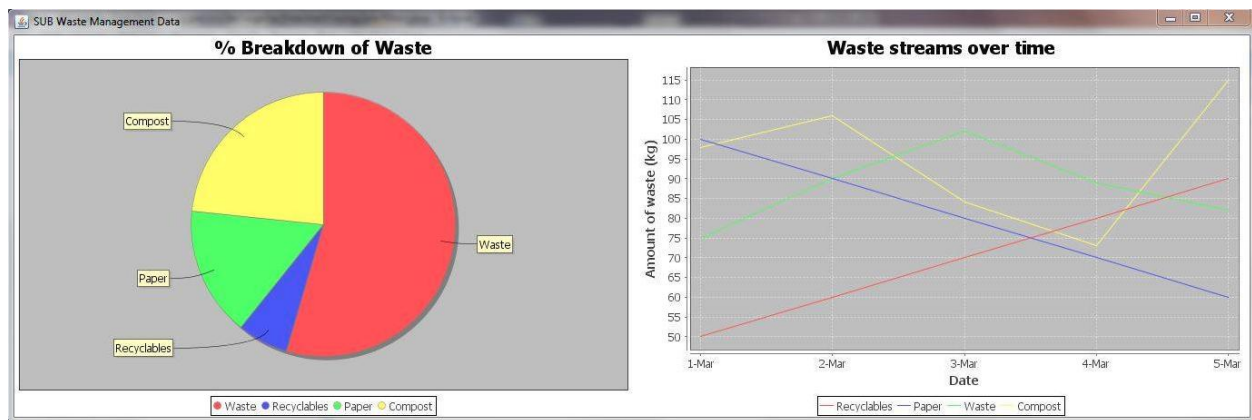


Figure 5. Example graphs created by the dashboard simulator

This application is a vertical prototype to show that it is possible to pull data off the server through the internet and display it in the form of graphs. Additional features such as selecting which streams to display and other graphs may be implemented in a stand-alone system. The final implementation for the client will not include this dashboard simulator program, as they will be hiring a company to do a more elaborate and customized dashboard system.

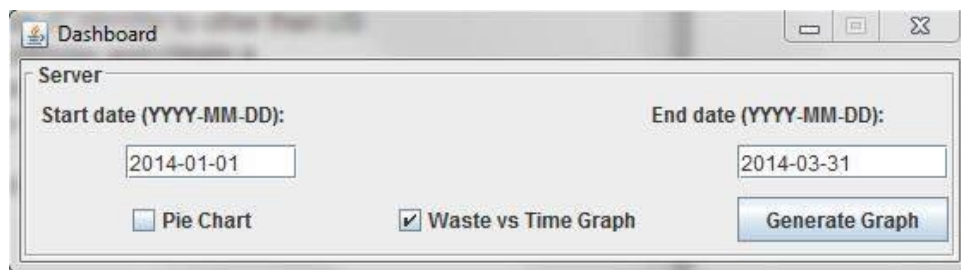


Figure 6. The graphical user interface for the dashboard simulator

4.0 Formal Testing

In order to verify the correctness of the software and hardware components a test plan is needed. As all of the software components require external I/O from either the hardware or through a serial port, it is difficult to write automated unit tests to verify the correctness of the software. As such, most of the testing suites consist of functional tests with manual user inputs.

The focus of the test plan for the computer application software is to verify that the application can correctly interface and handshake with the indicator and button panel, as well as the server. The test suite includes white-box test cases to verify that the correct actions are taken given certain serial port data received and combinations of user inputs. The test cases surrounding the user scenarios, are shown in Appendix C.

The test plan for the server is to ensure that a connection can be established to the server and data can be pushed and pulled. The test suite includes black-box test cases to ensure that each input triggers a correct output. The test cases are exhaustive therefore all possible cases are covered within these test cases. The test cases involving the server are shown in Appendix C.

The test plan for the dashboard application focuses on user functionality. The test suite includes black-box test cases to ensure that dates entered are valid and queries sent are correct. The test cases are conducted using manual user input and accounts for as many scenarios as possible.

The test suites for the computer application and button panel were conducted individually, while the components were stand-alone, as well as while collectively. This results in an integration test of sorts, and also allowed us to verify that the system would function correctly under actual operating conditions and user scenarios with the complete system.

5.0 Risk Management

The major forms of risk involved with our project include operational risks such as safety and hardware durability, and risk during project development. As the operational risks are most apparent to the users and to the client throughout the lifetime of the project, we have focused on minimizing and offering recommendations to minimize these risks.

The most important risk to take into consideration is the safety of our system. The scale and ramps are low-profile, and as such are a hazard for passersby. As a result, we have determined a safe placement for the scale and ramps, which is out of the main traffic pathway through the waste facility. The Anyload FSP floor scale is chosen over its alternatives because it fulfills the requirements of being durable and water-resistant to lower the risk of damage to it, while also being inexpensive. Checker plate is used for the ramps to provide extra traction, and side rails are used to prevent the waste bins from falling off the side of the ramps. The button panel is mounted to the wall at shoulder-height and is located right next to the scale to prevent people from having to reach to access the buttons.

In order to minimize the loss of data in the event of an outage or workstation failure, Dropbox is used as a form of version control. Source documents are uploaded and organized upon implementation of any major feature or milestone in order to back up a copy of functional code. Since there are only one or two people working on a given software component, a more advanced version control system such as Git is determined to not be necessary. Testing documents allow us to verify the correctness of our hardware and software, and also help us to predict potential user scenarios.

6.0 Client Hand-Off Protocol

The waste management system is currently installed in the SUB waste facility. Pending the installation of power and cabling and the procurement of the AMS server and computer, the final implementation can be installed. This includes the waste scale and ramps, indicator, button panel and software. The system will initially be used to only measure the weight of the waste bins and store the data locally and on the server, and once the electronic dashboard is designed the data on the server will be accessed.

In addition to the system implementation, our client will receive a short user manual in order to train users, as shown in Appendix D. The manual contains instructions for installation and maintenance, as well as operation instructions for both the janitorial staff operating the button panel and scale, and the IT staff in charge of running the software application. The focus of this user manual is for use as an aid for training the users to correctly operate the system. In addition, the basic specifications such as the capacity of the scale are included. The client will also receive complete documentation of the project, including the source code, testing documents, schematics and project reports.

7.0 Conclusion

A tested and stable waste management system is created, capable of recording the weight of waste bins, as well as classifying the waste into different waste streams through user input. The designed computer application can connect to a SQL server, interface with the indicator and button panel and save collected data online. The waste system is currently installed in the SUB, although the AMS server has not been created and the dashboard has not been designed yet.

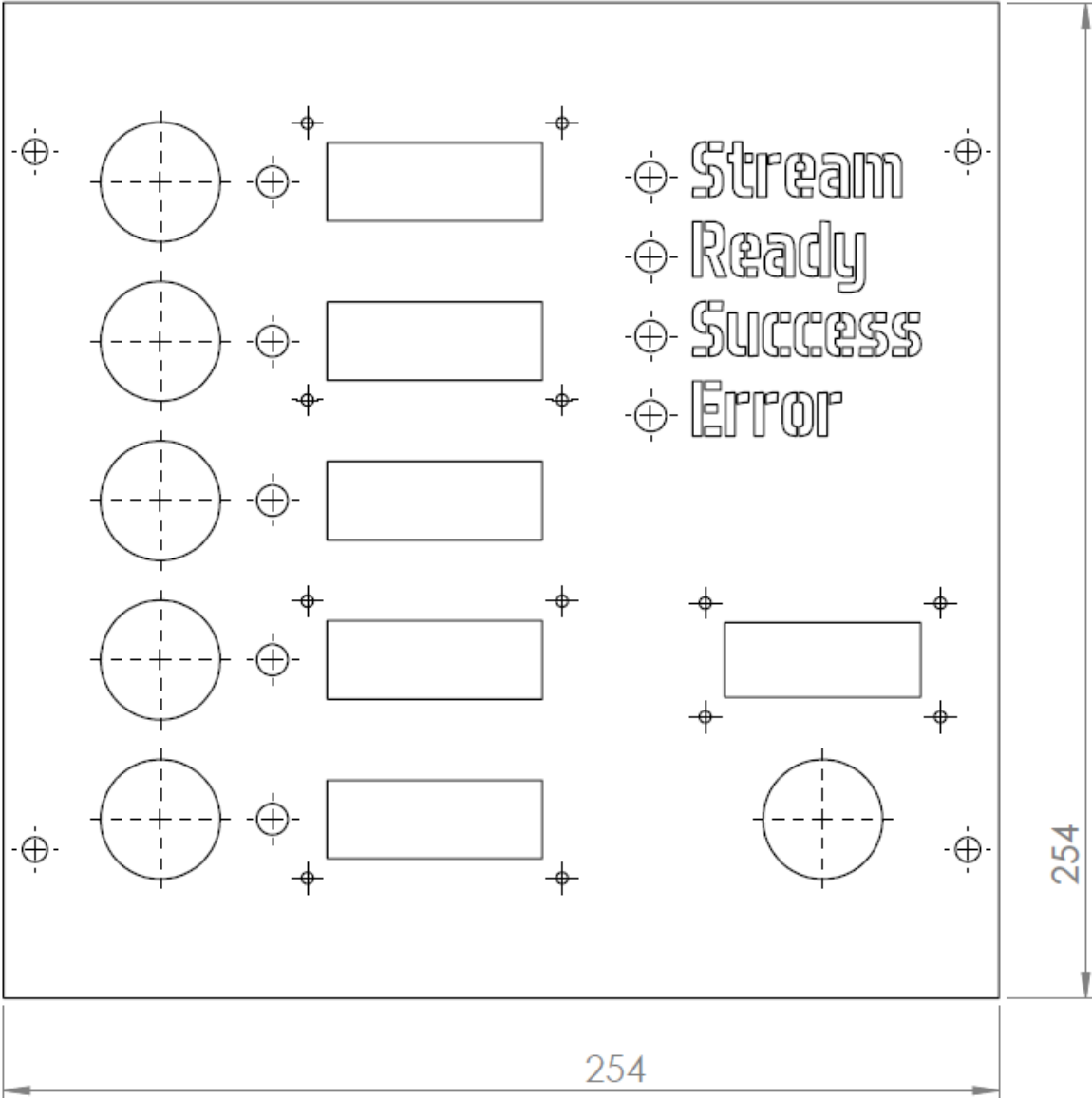
Our client will be delivered the necessary hardware and software needed to operate the waste scale. This includes the floor scale, indicator, button panel and the software application to run on a computer. Additionally, our client will receive a short user manual to help train the users, as well as complete documentation of the testing, source code and schematics.

Our ECE budget consisted of \$650, of which \$605.16 was spent. The majority of the ECE budget was spent on hardware and electrical components for the push button panel. In addition to our ECE budget, the AMS provided \$5000 of additional funding. Of the AMS funding, \$2951.20 of which was spent on purchasing the floor scale, the indicator and the two ramps leading up to the scale.

As the project will hopefully be ported to track waste at other buildings at UBC, it is likely that another group of students or engineers will expand on this project. It is recommended to research and try implementing a RS-485 or other serial port connection between the Arduino and the computer, as RS-232 is less reliable with longer cable lengths. A driver for the indicator and for the server can be created on the button panel microcontroller, effectively cutting out the computer. Placing all of the software on the button panel also increases the portability of the system, as the computer would no longer be necessary.

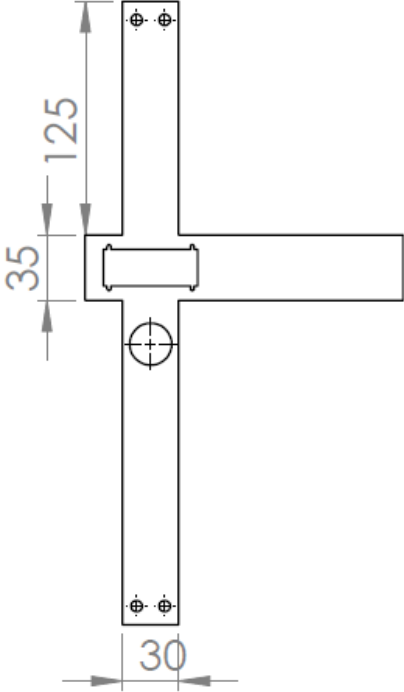
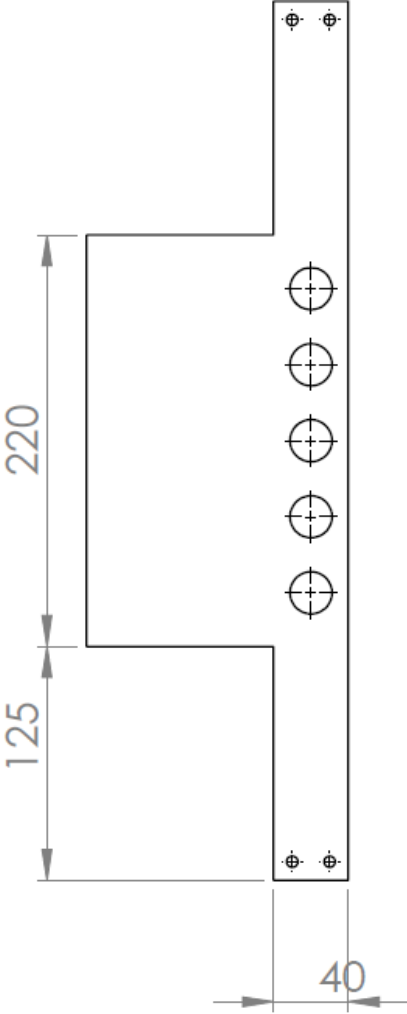
Appendix A: Drawings and Schematics

Button Panel cover drawing (measurements in millimetres):

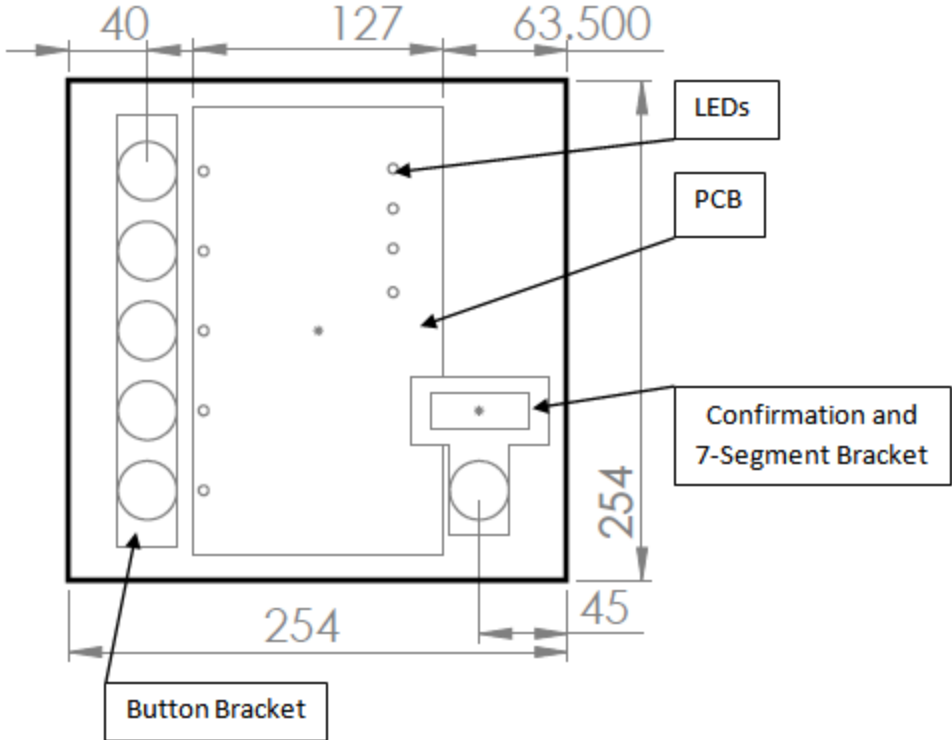


Push Button Bracket (left) and Confirmation and 7-Segment Bracket (right) drawings

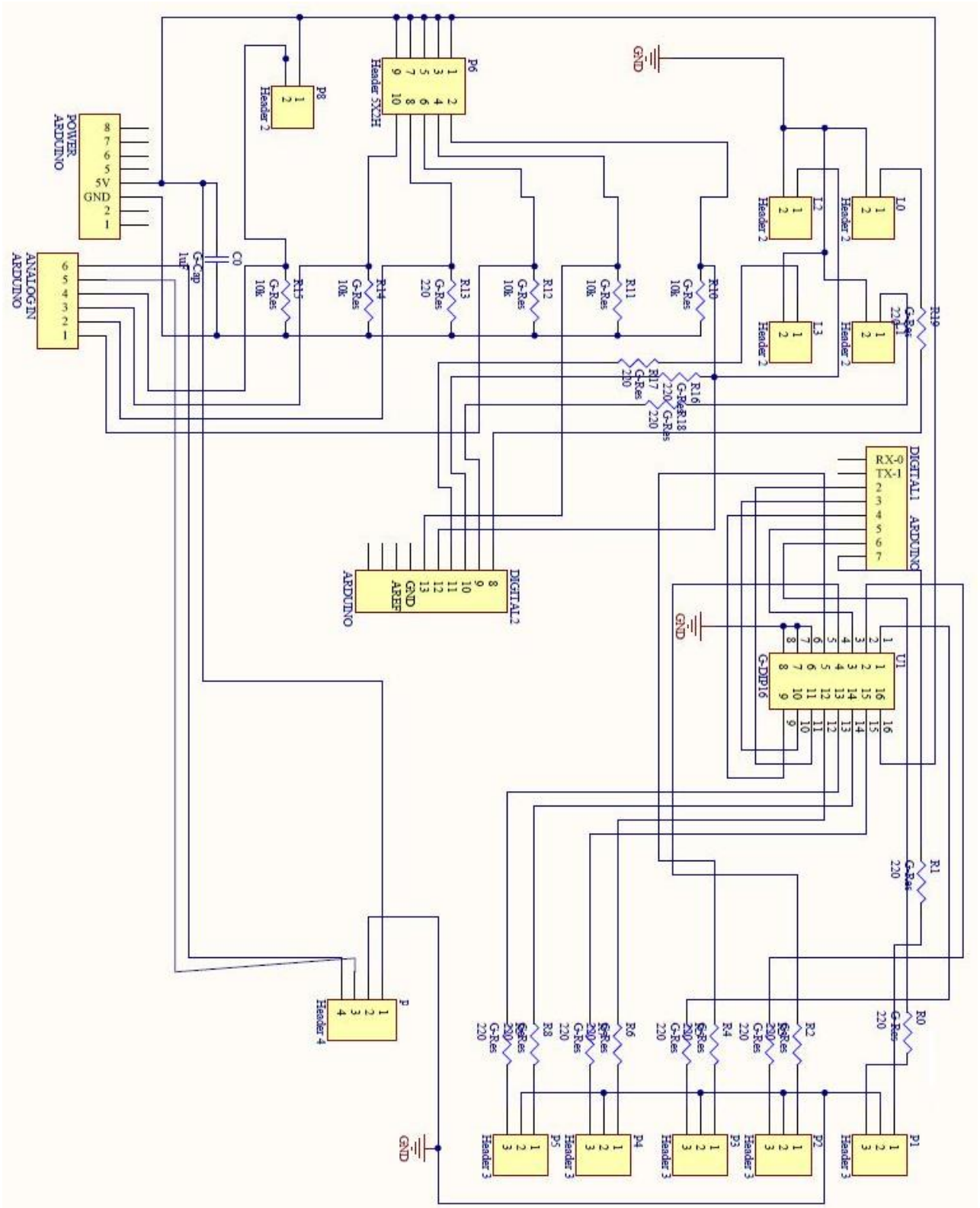
(measurements in millimetres):



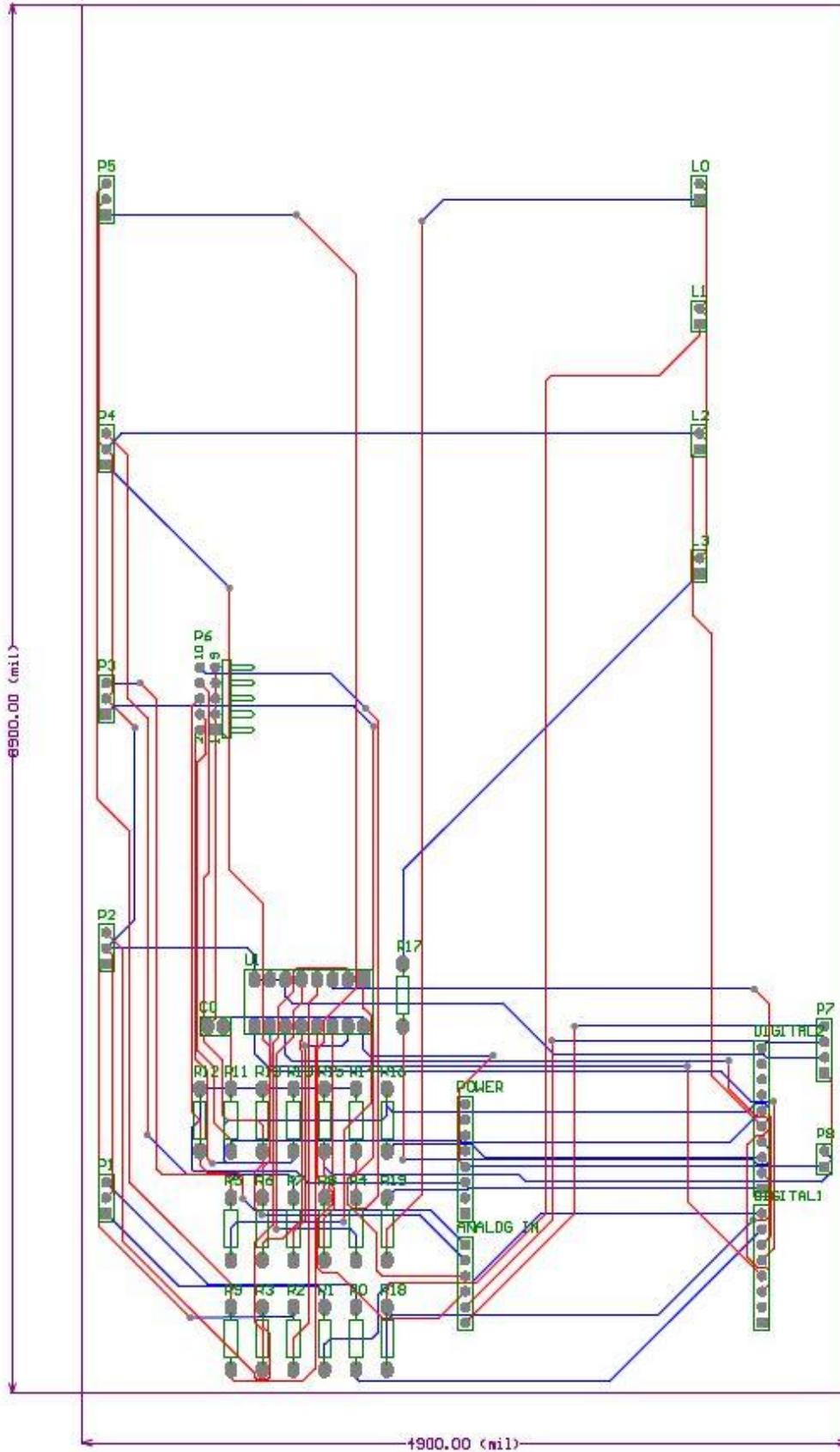
Button Panel Box layout drawing (measurements in millimetres):



Button Panel Circuit Schematic:



PCB Schematic:



Appendix B: Source Code

Java Application

Control Driver

```
package scale;

import gnu.io.SerialPortEvent;

public class ControlDriver extends SerialDriver {
    private Object controlLock = new Object();
    private IndicatorDriver indicatorDriver;
    private byte controlData;
    private boolean initialized = false;
    private boolean[] streamEnable = new boolean[5];
    private boolean[] tareEnable = new boolean[5];
    private String[] streams = new String[5];
    private float[] tares = new float[10];

    // Purpose: Event listener for the serial port, used for receiving and parsing data
    // Input: event: The serial port event
    @Override
    public void serialEvent(SerialPortEvent event) {
        synchronized(controlLock) {
            // Check if data is available
            if(event.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
                try {
                    controlData = (byte)input.read();

                    //
                    System.out.println("Data read: \" + controlData + "\"");

                    switch(controlData) {
                        case '0': // Set the stream to Stream 0, send confirmation to
                                control panel

                                if(streamEnable[0]) {
                                    // Check if the stream was already selected
                                    if(indicatorDriver.getStream() == 0 &&
                                        tareEnable[0]) {
                                        if(indicatorDriver.getTare() == 0) {
                                            indicatorDriver.setTare(5);
                                            sendData("T01\n");
                                        } else {
                                            indicatorDriver.setTare(0);
                                            sendData("T00\n");
                                        }
                                    } else {
                                        // Another stream was selected,
                                        select this one
                                        indicatorDriver.setStream(0);
                                        indicatorDriver.setTare(0);
                                        sendData("T00\n");
                                    }
                                } else {
                                    // Error: Stream is not enabled
                                    sendData("E\n");
                                    indicatorDriver.setStream(-1);
                                    indicatorDriver.setTare(-1);
                                }
                            }
                }
            }
        }
    }
}
```

```

    }
    break;
case '1': // Set the stream to Stream 1, send confirmation to
control panel
    if(streamEnable[1]) {
    // Check if the stream was already selected
    if(indicatorDriver.getStream() == 1 &&
tareEnable[1]) {
        if(indicatorDriver.getTare() == 1) {
            sendData("T11\n");
        } else {
            sendData("T10\n");
        }
    } else {
        // Another stream was selected,
        indicatorDriver.setStream(1);
        indicatorDriver.setTare(1);
        sendData("T10\n");
    }
    } else {
    // Error: Stream is not enabled
    sendData("E\n");
    indicatorDriver.setStream(-1);
    indicatorDriver.setTare(-1);
    }
    break;
case '2': // Set the stream to Stream 2, send confirmation to
control panel
    if(streamEnable[2]) {
    // Check if the stream was already selected
    if(indicatorDriver.getStream() == 2 &&
tareEnable[2]) {
        if(indicatorDriver.getTare() == 2) {
            sendData("T21\n");
        } else {
            sendData("T20\n");
        }
    } else {
        // Another stream was selected,
        indicatorDriver.setStream(2);
        indicatorDriver.setTare(2);
        sendData("T20\n");
    }
    } else {
    // Error: Stream is not enabled
    sendData("E\n");
    indicatorDriver.setStream(-1);
    indicatorDriver.setTare(-1);
    }
    break;
case '3': // Set the stream to Stream 3, send confirmation to
control panel

```



```
// Purpose: Gets the stream, given the index number
// Input: i: The index of the stream array
// Output: The name of the stream
public String getStream(int i) {
    return streams[i];
}

// Purpose: Gets the tare, given the index number
// Input: i: The index of the tare array
// Output: The tare value
public float getTare(int i) {
    return tares[i];
}

// Purpose: Checks if the button panel was successfully initialized
// Output: The value of the initialized variable
public boolean getInitialized() {
    return initialized;
}

// Purpose: Gets a references to the IndicatorDriver so that the ControlDriver
//         can call methods from the IndicatorDriver (i.e. to send weight
//         requests)
// Input: indicatorDriver: The IndicatorDriver object to reference
public void withIndicatorPort(IndicatorDriver indicatorDriver) {
    this.indicatorDriver = indicatorDriver;
}
}
```

Indicator Driver

```
package scale;

import gnu.io.SerialPortEvent;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.Connection;
import java.sql.Date;
import java.sql.Time;
import java.text.DecimalFormat;
import java.util.GregorianCalendar;
import java.util.Timer;
import java.util.TimerTask;

import server.WasteTable;

public class IndicatorDriver extends SerialDriver {
    private DecimalFormat d = new DecimalFormat("#.##");
    private ControlDriver controlDriver;
    private Connection server;
    private WasteTable table;
    private GregorianCalendar calendar;
    private Object indicatorLock = new Object();
    private String indicatorData = new String();
    private String directoryPath;
    private String path;
    private BufferedWriter fileWriter;
    private BufferedReader fileReader;
    private int currentStream = -1; // The index of the currently selected stream
    private int currentTare = -1; // The index of the currently selected tare

    private float tare = 0;
    private float gross;
    private float weight;
    private String units;
    private String stream;
    private int mode; // 0 = gross, -1 = error
    private int status; // 0 = none, 1 = centre of zero, 2 = below zero, 3 = over capacity, -1 = error
    private Date date;
    private Time time;
    private boolean confirmation; // Determines whether the weight sample will be saved
    private final int pollRate = 500; // Weight polling rate in milliseconds

    // Purpose: Event listener for the serial port, used for receiving and parsing data
    // Input: event: The serial port event
    @Override
    public void serialEvent(SerialPortEvent event) {
        synchronized(indicatorLock) {
            // Check if data is available
            if(event.getEventType() == SerialPortEvent.DATA_AVAILABLE) {
                try {
                    // Save the data byte-by-byte to an array
                    indicatorData = input.readLine();
                }
            }
        }
    }
}
```

```

// Make sure the user didn't press the Print button on the indicator
if(indicatorData.contains("Weight ticket. ")) {
    System.out.println("Error: Do not press the Print button on the
indicator");

    // Discard the next 4 lines -- these are part of the weight ticket
    input.readLine();
    input.readLine();
    input.readLine();
    input.readLine();

    controlDriver.sendData("E\n");
}
// Check if the weight was over capacity
if(indicatorData.contains("OCAP")) {
    controlDriver.sendData("WO\n");
}
// Check the length to make sure the data is of the correct format
if(indicatorData.length() == 17) {
    // Parse the weight data here
    System.out.println("Data: \" + indicatorData + "\"");

    // Check if this sample will be saved (assuming everything is
    alright)

    if(confirmation) {

        // Reset the confirmation flag
        confirmation = false;

        // Parse the weight sample
        int result = parseWeight(indicatorData);

        // Print the weight sample to the log
        printWeight();

        if(result == 0) {
            if(table != null) {
                // Send the data to the server
                table.insertWaste(server, weight,

stream, date, time);

            }

            // Write to the file
            String fileName = date.toString().substring(0,

10) + ".txt";

            writeToFile(fileName, this.toString());

            controlDriver.sendData("S\n");
        } else {
            controlDriver.sendData("E\n");
        }

        setStream(-1);
        setTare(-1);
    } else {
        // Construct a weight sample message for the Arduino
        String message = "W";

        if(indicatorData.substring(14, 16).equals("OC")) {
            message += "O";
        } else {
            if(indicatorData.substring(14,

16).equals("MO")) {

```



```

        message += "U";
    } else {
        message += "S";
    }
    if(indicatorData.contains("-")) {
        message += ("- " +
indicatorData.substring(4, 8) + "\n");
    } else {
        message +=
(indicatorData.substring(3, 8) + "\n");
    }
}
controlDriver.sendData(message);
}
} catch(Exception e) {
    e.printStackTrace();
}
}
}

// Purpose: Gets a references to the ControlDriver so that the IndicatorDriver
// can call methods from the ControlDriver (i.e. to send confirmation
// messages)
// Input: controlDriver: The ControlDriver object to reference
public void withControlPort(ControlDriver controlDriver) {
    this.controlDriver = controlDriver;
}

// Purpose: Gets a reference to the server connection
// Input: server: The Connection object to reference
// table: The WasteTable object to reference
public void withServer(Connection server, WasteTable table) {
    this.server = server;
    this.table = table;
}

// Purpose: Parses the string of data received from the indicator into weight,
// units, time and status bits
// Input: data: The string of data to parse
// Output: 0 on success, 6 if unsuccessful
private int parseWeight(String data) {
    synchronized(indicatorLock) {
        int polarity;

        if(data.length() == 17) {
            // Determine the polarity of the weight
            if(data.charAt(0) == '-') {
                polarity = -1;
            } else {
                polarity = 1;
            }

            // Parse the weight
            for(int i = 1; i < 8; i++) {
                // Ignore the spaces, find when the digits start
                if(data.charAt(i) != ' ') {
                    gross = polarity * Float.valueOf(data.substring(i, 8));
                    break;
                }
            }
        }
    }
}

```

```

    }

    weight = Float.valueOf(d.format(gross - tare));

    // Check that the weight is positive
    if(weight <= 0) {
        System.out.println("Error: Net weight is negative");
        return 6;
    }

    // Parse the units
    units = data.substring(9, 11).toLowerCase();
    if(!units.equals("kg") && !units.equals("lb") && !units.equals("oz") &&
!units.equals(" g")) {
        System.out.println("Error: Invalid unit of measurement");
        return 6;
    }

    // Parse the mode
    if(data.charAt(12) == 'G') {
        mode = 0;
    } else {
        mode = -1;
        System.out.println("Error: Expected gross weight measurement");
        return 6;
    }

    // Parse the status bits
    switch(data.substring(14, 16)) {
        case " ": status = 0;
            break;
        case "CZ": status = 1;
            return 6;
        case "BZ": status = 2;
            return 6;
        case "MO": status = 3;
            return 6;
        case "OC": status = 4;
            return 6;
        default: status = -1;
            System.out.println("Error: Invalid status bits");
            return 6;
    }

    // Get the date
    calendar = new GregorianCalendar();
    time = new Time(calendar.getTime().getTime());
    date = new Date(calendar.getTime().getTime());

    return 0;
} else {
    System.out.println("Error: Expected a 17-character string");
    return 6;
}
}

// Purpose: Prints a ticket containing all of the weight data collected
// on the console
public void printWeight() {
    synchronized(indicatorLock) {
        if(mode != 0) {

```

```

        System.out.println("The indicator was not in gross measurement mode");
    }

    switch(status) {
        case 0: System.out.println(this.toString());
                break;
        case 1: System.out.println("The weight was too close to 0");
                break;
        case 2: System.out.println("The weight was negative");
                break;
        case 3: System.out.println("The weight was taken while the load was unstable");
                break;
        case 4: System.out.println("The weight was over capacity");
                break;
        default: System.out.println("There was some error");
                break;
    }
}

// Purpose: Sets the confirmation flag, so that the next weight sample will
// be saved
public void setConfirmation() {
    synchronized(indicatorLock) {
        confirmation = true;
    }
}

// Purpose: Sets the waste stream type
// Input: i: The index of the stream array that is currently selected, -1 if null
public void setStream(int i) {
    synchronized(indicatorLock) {
        if(i >= 0) {
            this.stream = controlDriver.getStream(i);
            this.currentStream = i;
        } else {
            currentStream = -1;
            this.stream = null;
        }
    }
}

// Purpose: Gets the waste stream type
// Output: The currently selected waste stream
public int getStream() {
    return currentStream;
}

// Purpose: Sets the tare value
// Input: i: The index of the tare array that is currently selected
public void setTare(int i) {
    synchronized(indicatorLock) {
        if(i >= 0) {
            this.tare = controlDriver.getTare(i);
            this.currentTare = i;
        } else {
            this.tare = 0;
            this.currentTare = -1;
        }
    }
}

```

```

// Purpose: Gets the tare value
// Output: The currently selected tare
public int getTare() {
    return currentTare;
}

// Purpose: Returns a string containing all of the data collected of the format
//      "date weight units stream"
// Output: The string representation of the weight sample
public String toString() {
    synchronized(indicatorLock) {
        return date.toString() + " " + time.toString() + " " + weight + " " + units + " " + stream;
    }
}

// Purpose: Sets the weight data log file directory (to the folder).
// Input: path: The absolute path of the folder containing the logs
public void setPath(String path) {
    synchronized(indicatorLock) {
        this.directoryPath = path;
    }
}

// Purpose: Checks whether fileName exists, then writes data to the end of
//      the file if it does exist, then closes the file
// Input: fileName: The name of the file, will be appended to the folder path
//      data: The data to write to the file
public void writeToFile(String fileName, String data) {
    synchronized(indicatorLock) {
        try {
            // Get the file path from the directory and the file name
            path = directoryPath + "\\ " + fileName;

            // Open the file in append mode if it exists, otherwise create a new file
            fileWriter = new BufferedWriter(new FileWriter(path, true));

            // Write the weight data to the file
            fileWriter.write(data);
            fileWriter.newLine();

            // Close the file
            fileWriter.close();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }
}

// Purpose: Writes the current data to the server, provided the weight is stable
public void writeToServer() {
    synchronized(indicatorLock) {
        table.insertWaste(server, weight, stream, date, time);
    }
}

// Purpose: Reads a line of text from the given file, if it exists, otherwise returns
//      a null string
// Output: The line of text, or null if the file does not exist
public String readLine(String fileName) {
    synchronized(indicatorLock) {
        String result = null;

```

```

    try {
        // Get the file path from the directory and the file name
        path = directoryPath + "\\\" + fileName;

        // Open the file in append mode if it exists, otherwise create a new file
        fileReader = new BufferedReader(new FileReader(path));

        // Read the line of text
        result = fileReader.readLine();

        // Close the file
        fileWriter.close();
    } catch(IOException e) {
        e.printStackTrace();
    }

    return result;
}

// Purpose: Sets up the timer to read the weight every 1s
public void setUpTimer() {
    new Timer().schedule(new TimerTask() {
        public void run() {
            sendData((byte)5);
        }
    }, 1000, pollRate);
}
}

```

MainUI

```
public class MainUI {
    public static void main(String[] args) {
        IndicatorDriver indicator = new IndicatorDriver();
        ControlDriver control = new ControlDriver();

        // Attach the indicator driver to the control driver and vice-versa
        control.withIndicatorPort(indicator);
        indicator.withControlPort(control);

        // Connect to the COM ports
        indicator.connect("COM9");
        control.connect("COM10");

        // Create a new thread
        Thread t = new Thread() {
            public void run() {
                while(true);
            }
        };
        t.start();
    }
}
```

Serial Driver

```
package scale;

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

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;

public class SerialDriver implements SerialPortEventListener{

    SerialPort serialPort;
    protected BufferedReader input;
    protected OutputStream output;

    // Purpose: Attempts to connect to a serial port, given the port name.
    // Also creates a SerialReader and SerialWriter thread for reading/writing
    // Input: portName - The COM port as a string (Eg. COM1)
    // Output: 0 on success, -1 if port does not exist, -2 if port is in use
    public int connect(String portName) {
        try {
            // Parse the port name
            CommPortIdentifier portIdentifier = CommPortIdentifier.getPortIdentifier(portName);

            // Check if the port is in use
            if(portIdentifier.isCurrentlyOwned()) {
                System.out.println("Error: " + portName + " is already in use.");
                return -2;
            } else {
                CommPort commPort = portIdentifier.open(this.getClass().getName(), 2000);

                // Check that it is a serial port
                if(commPort instanceof SerialPort) {
                    // Set the serial port to 8 data bits, 1 stop bit, no parity, 9600 baud
                    SerialPort serialPort = (SerialPort) commPort;
                    serialPort.setSerialPortParams(9600, SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);

                    // Create the input and output streams
                    input = new BufferedReader(new
                    InputStreamReader(serialPort.getInputStream()));
                    output = serialPort.getOutputStream();

                    // Add the event listener
                    serialPort.addEventListener(this);
                    serialPort.notifyOnDataAvailable(true);

                    return 0;
                } else {
                    System.out.println("Error: " + portName + " is not a serial port");
                    return -1;
                }
            }
        } catch(Exception e) {
            return -1;
        }
    }
}
```

```

    }
}

// Purpose: Closes the serial port to avoid port locking from other applications
public synchronized void close() {
    if(serialPort != null) {
        serialPort.removeEventListener();
        serialPort.close();
    }
}

// Purpose: Writes a string to the serial port to send
// Input: data: The string to send
public synchronized void sendData(String data) {
    try {
        output.write(data.getBytes());
    } catch(IOException e) {
        e.printStackTrace();
    }
}

// Purpose: Writes a byte of data to the serial port to send
// Input: data: The byte to send
public synchronized void sendData(byte data) {
    try {
        output.write(data);
    } catch(IOException e) {
        e.printStackTrace();
    }
}

// Purpose: Event listener for the serial port, used for receiving and parsing data
// Input: event: The serial port event
public synchronized void serialEvent(SerialPortEvent event) {
    // Overridden in the child classes
}
}

```


WasteTable

```
package server;

import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Time;

public class WasteTable {

    private static final String CreateTableWasteManagement = "CREATE TABLE WasteStreamManagement" +
        "(WasteStream CHAR(20)," + "dDay DATE," + "dTime TIME(0)," + "weightnum float(2)," + "WID INT IDENTITY(1,1)" +
        "NOT NULL," + "PRIMARY KEY(WID))";
    private static final String DropTableWasteManagement = "DROP TABLE WasteStreamManagement";
    //private static final String createSequence = "CREATE SEQUENCE WIDCounter\n" + "START WITH 1\n" +
    "INCREMENT BY 1";
    public void createWaste(Connection con) throws SQLException {

        Statement state = null;
        state = con.createStatement();
        state.executeUpdate(CreateTableWasteManagement);
    //    state.executeUpdate(createSequence);
        con.commit();
    }
    public void DropTableWaste(Connection con)
    {
        Statement state = null;
        try{
            state = con.createStatement();
            state.executeUpdate(DropTableWasteManagement);

        }catch(SQLException e){
            e.printStackTrace();
        }
    }

    public void showWasteTable (Connection connect){
        Statement state;
        ResultSet r;

        try{
            state = connect.createStatement();
            r = state.executeQuery("SELECT * FROM WasteStreamManagement");

            while(r.next()){
                String entry = r.getInt("WID") + " " + r.getDate("dDay").toString() + " " +
                    r.getTime("dTime").toString() + " " +
                        r.getFloat("weightnum") + " " + r.getString("WasteStream").trim();

                System.out.println(entry);
            }

        }catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```

public void insertWaste (Connection con, float number, String wastestream, Date day, Time time)
{
    PreparedStatement p;
    float weight = number;
    String Waste = wastestream;

    try
    {
        p = con.prepareStatement("INSERT INTO WasteStreamManagement VALUES (?, ?, ?, ?)");

        //GregorianCalendar calendar = new GregorianCalendar();

        //java.sql.Date daytime = new java.sql.Date(calendar.getTime().getTime());
        p.setString(1, Waste);
        p.setDate(2, day);
        p.setTime(3, time);
        p.setFloat(4, weight);

        p.executeUpdate();
        con.commit();
        p.close();
    } catch(SQLException error) {
        System.out.println ("INSERTION ERROR \n");
        error.printStackTrace();
    }
}

public void deleteWaste (Connection con, int Date)
{
}

}

```

LogUI

```
package ui;

import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Rectangle;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintStream;
import java.sql.Date;
import java.sql.Time;
import java.util.GregorianCalendar;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;

public class LogUI extends JPanel {
    private static final long serialVersionUID = 1L;

    private JTextArea textArea;
    private JFrame frame;
    private String fileDirectory = null;

    public LogUI() {
        frame = new JFrame("Log");

        textArea = new JTextArea(20, 40);
        textArea.setEditable(false);
        JScrollPane scrollBar = new JScrollPane(textArea);

        // Layout components using the GridBag layout manager
        new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();

        c.gridwidth = GridBagConstraints.REMAINDER;
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        c.weighty = 1.0;
        add(scrollBar, c);

        // Set the application to be killed when the window is closed
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Dimension d = frame.getToolkit().getScreenSize();
        Rectangle r = frame.getBounds();
        frame.setLocation((d.width - r.width) / 2, (d.height - r.height) / 2);

        frame.add(this);
        frame.pack();
    }

    public void showLog() {
        frame.setVisible(true);
    }
}
```

```

public JFrame getFrame() {
    return frame;
}

public void setFileDirectory(String path) {
    fileDirectory = path;
}

private void updateTextArea(final String text) {
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            String textToWrite = new String();

            if(!text.contains("\n")) {
                GregorianCalendar calendar = new GregorianCalendar();
                Time time = new Time(calendar.getTime().getTime());
                Date date = new Date(calendar.getTime().getTime());

                textToWrite = textToWrite.concat "[" + date.toString() + " - " +
time.toString() + "]" );
            }
            textToWrite = textToWrite.concat(text);

            textArea.append(textToWrite);

            // Print the log outputs to a file also
            if(fileDirectory != null) {
                try {
                    BufferedWriter fileWriter = new BufferedWriter(new
FileWriter(fileDirectory + "\\log.txt", true));

                    fileWriter.append(textToWrite);
                    fileWriter.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    });
}

public void redirectSystemOutput() {
    OutputStream output = new OutputStream() {
        @Override
        public void write(int b) throws IOException {
            updateTextArea(String.valueOf((char)b));
        }

        @Override
        public void write(byte[] b, int off, int len) throws IOException {
            updateTextArea(new String(b, off, len));
        }

        @Override
        public void write(byte[] b) throws IOException {
            updateTextArea(new String(b));
        }
    };

    System.setOut(new PrintStream(output, true));
}
}

```

MainUI

```
package ui;

public class MainUI {
    public static void main(String[] args) {
        // Instantiate the server login GUI and server connection, refresh connection status
        // every second
        StartupUI startupUI = new StartupUI();
        while(!startupUI.checkConnection()) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // Create a new thread
        Thread t = new Thread() {
            public void run() {
                // Everything is done! Print a confirmation message to the log
                System.out.println("Ready");

                while(true);
            }
        };
        t.start();
    }
}
```

StartupUI

```
package ui;

import gnu.io.CommPortIdentifier;

import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Insets;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;

import scale.ControlDriver;
import scale.IndicatorDriver;
import server.WasteTable;

public class StartupUI implements ActionListener, ItemListener {
    // Login variables
    private Connection connection;
    private boolean connected;
    private String settingsPath;
    private String indicatorPort;
    private String controlPort;
    private String fileDirectory;
    private String username;
    private String databaseName;
```

```

private String port;
private String ipAddress;
private float[] tares = new float[10];

// User is allowed 3 login attempts or else the program will close
private int loginAttempts = 0;

// Server settings fields
private JLabel portLabel = new JLabel("Port:");
private JLabel ipAddressLabel = new JLabel("IP Address:");
private JLabel databaseNameLabel = new JLabel("Database Name:");
private JLabel usernameLabel = new JLabel("Username:");
private JLabel passwordLabel = new JLabel("Password:");
private JTextField portField;
private JTextField ipAddressField;
private JTextField databaseNameField;
private JTextField usernameField;
private JPasswordField passwordField;

// File system directory fields
private JLabel directoryLabel = new JLabel("File Directory:");
private JTextField directoryField;
private JButton browseButton;
private JFileChooser fileChooser;

// Com port settings fields
private JLabel indicatorLabel = new JLabel("Scale:");
private JLabel controlLabel = new JLabel("Button Panel:");
private JComboBox<String> indicatorField;
private JComboBox<String> controlField;
private JButton refreshButton;

// Waste stream settings fields
private JCheckBox stream1Enable;
private JCheckBox stream2Enable;
private JCheckBox stream3Enable;
private JCheckBox stream4Enable;
private JCheckBox stream5Enable;
private JCheckBox tare1aEnable;
private JCheckBox tare2aEnable;
private JCheckBox tare3aEnable;
private JCheckBox tare4aEnable;
private JCheckBox tare5aEnable;
private JLabel stream1Label = new JLabel("Stream 1:");
private JLabel stream2Label = new JLabel("Stream 2:");
private JLabel stream3Label = new JLabel("Stream 3:");
private JLabel stream4Label = new JLabel("Stream 4:");
private JLabel stream5Label = new JLabel("Stream 5:");
private JLabel tare1Label = new JLabel("Tare:");
private JLabel tare2Label = new JLabel("Tare:");
private JLabel tare3Label = new JLabel("Tare:");
private JLabel tare4Label = new JLabel("Tare:");
private JLabel tare5Label = new JLabel("Tare:");
private JLabel tare1aLabel = new JLabel("Enable second tare");
private JLabel tare2aLabel = new JLabel("Enable second tare");
private JLabel tare3aLabel = new JLabel("Enable second tare");
private JLabel tare4aLabel = new JLabel("Enable second tare");
private JLabel tare5aLabel = new JLabel("Enable second tare");
private JLabel tare1UnitsLabel = new JLabel("kg");
private JLabel tare2UnitsLabel = new JLabel("kg");
private JLabel tare3UnitsLabel = new JLabel("kg");
private JLabel tare4UnitsLabel = new JLabel("kg");

```

```

private JLabel tare5UnitsLabel = new JLabel("kg");
private JLabel tare1aUnitsLabel = new JLabel("kg");
private JLabel tare2aUnitsLabel = new JLabel("kg");
private JLabel tare3aUnitsLabel = new JLabel("kg");
private JLabel tare4aUnitsLabel = new JLabel("kg");
private JLabel tare5aUnitsLabel = new JLabel("kg");
private JTextField stream1Field;
private JTextField stream2Field;
private JTextField stream3Field;
private JTextField stream4Field;
private JTextField stream5Field;
private JTextField[] tareFields = new JTextField[10];

// Startup buttons
private JButton startButton;
private JButton offlineModeButton;
private JButton saveButton;
private JButton creditsButton;
private JFrame mainFrame;
private JPanel mainPanel;
private LogUI logUI;

// Credits
private String credits =
    "UBC Digital Waste Management System v1.0\n" +
    "UBC Electrical and Computer Engineering Capstone Design Course\n" +
    "September 2013 - April 2014\n\n" +
    "Designed by:\n" +
    " - Brandon Lowe\n" +
    " - Brian Yim\n" +
    " - Kyle Lee\n" +
    " - Luis Gan\n\n" +
    "Special thanks to:\n" +
    " - The Java RXTX serial port library\n" +
    " - Adafruit LED matrix libraries\n" +
    " - Leo Stocco, Capstone instructor\n" +
    " - Maan Almarghalani, Capstone teaching assistant\n"
    ;

// Constructor
public StartupUI() {
    mainFrame = new JFrame("Capstone Waste Scale");
    mainFrame.setResizable(false);
    mainPanel = new JPanel();
    mainPanel.setLayout(new BorderLayout(mainPanel, BorderLayout.PAGE_AXIS));
    mainFrame.setContentPane(mainPanel);

    // Open up the log UI
    logUI = new LogUI();
    logUI.redirectSystemOutput();

    // Create the server panel
    createServerUI();

    // Create the file directory panel
    createDirectoryUI();

    // Create the serial port panel
    createSerialPortUI();

    // Create the stream settings panel
    createStreamUI();

```



```

// Create the start up panel
createStartupUI();

// Open the settings file
openSettings();

// Anonymous inner class for closing the window
mainFrame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});

// size the window to obtain a best fit for the components
mainFrame.pack();

// center the frame
Dimension d = mainFrame.getToolkit().getScreenSize();
Rectangle r = mainFrame.getBounds();
mainFrame.setLocation((d.width - r.width) / 2,
(d.height - r.height) / 2);

// make the window visible
mainFrame.setVisible(true);

// place the cursor in the text field for the username
usernameField.requestFocus();

try {
    // Load the Oracle JDBC driver
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
} catch(Exception e) {
    e.printStackTrace();
    System.exit(-1);
}

}

// Purpose: Creates the server settings section of the GUI
private void createServerUI() {
    GridBagLayout g = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    c.fill = GridBagConstraints.HORIZONTAL;
    JPanel serverPanel = new JPanel(g);

    mainPanel.add(serverPanel);

    // Create the border
    TitledBorder title =
BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED), "Server");
    title.setTitleJustification(TitledBorder.LEFT);
    serverPanel.setBorder(title);

    // Add in the username field
    c.gridx = 0;
    c.gridy = 0;
    c.insets = new Insets(5,5,5,0);
    serverPanel.add(usernameLabel, c);
    usernameField = new JTextField(10);
    c.gridx = 1;
    c.gridy = 0;
    c.insets = new Insets(5,5,5,0);

```

```

serverPanel.add(usernameField, c);

// Add in the IP address field
c.gridx = 2;
c.gridy = 0;
c.insets = new Insets(5,20,5,0);
serverPanel.add(ipAddressLabel, c);
ipAddressField = new JTextField(10);
c.gridx = 3;
c.gridy = 0;
c.gridwidth = 2;
c.insets = new Insets(5,5,5,0);
serverPanel.add(ipAddressField, c);

// Add in the server port field
c.gridx = 5;
c.gridy = 0;
c.gridwidth = 1;
c.insets = new Insets(5,20,5,0);
serverPanel.add(portLabel, c);
portField = new JTextField(5);
c.gridx = 6;
c.gridy = 0;
c.insets = new Insets(5,5,5,5);
serverPanel.add(portField, c);

// Add in the password field
c.gridx = 0;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
serverPanel.add(passwordLabel, c);
passwordField = new JPasswordField(10);
passwordField.setEchoChar('*');
c.gridx = 1;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
serverPanel.add(passwordField, c);

// Add in the database name field
c.gridx = 2;
c.gridy = 1;
c.gridwidth = 2;
c.insets = new Insets(5,20,5,0);
serverPanel.add(databaseNameLabel, c);
databaseNameField = new JTextField(15);
c.gridx = 4;
c.gridy = 1;
c.gridwidth = 3;
c.insets = new Insets(5,5,5,5);
serverPanel.add(databaseNameField, c);
}

// Purpose: Creates the serial port settings section of the GUI
private void createSerialPortUI() {
    GridBagLayout g = new GridBagLayout();
    GridBagConstraints c = new GridBagConstraints();
    JPanel serialPanel = new JPanel(g);

    mainPanel.add(serialPanel);

    // Create the border

```

```

        TitledBorder title =
BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED), "Serial Ports");
        title.setTitleJustification(TitledBorder.LEFT);
        serialPanel.setBorder(title);

        // Get the list of serial ports
        List<String> list = getAvailableSerialPorts();
        String[] portList = list.toArray(new String[list.size()]);

        indicatorField = new JComboBox<String>(portList);
        indicatorField.setSelectedIndex(0);
        indicatorField.addActionListener(this);

        controlField = new JComboBox<String>(portList);
        controlField.setSelectedIndex(0);
        controlField.addActionListener(this);

        // Add the indicator port field
        c.gridx = 0;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        serialPanel.add(indicatorLabel, c);
        c.gridx = 1;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        serialPanel.add(indicatorField, c);

        // Add the button panel port field
        c.gridx = 2;
        c.gridy = 0;
        c.insets = new Insets(5,20,5,0);
        serialPanel.add(controlLabel, c);
        c.gridx = 3;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        serialPanel.add(controlField, c);

        // Add the refresh button
        refreshButton = new JButton("Refresh Available Ports");
        refreshButton.addActionListener(this);
        c.gridx = 4;
        c.gridy = 0;
        c.insets = new Insets(5,10,5,5);
        serialPanel.add(refreshButton, c);
    }

    // Purpose: Creates the file directory section of the GUI
    private void createDirectoryUI() {
        GridBagLayout g = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        JPanel filePanel = new JPanel(g);

        mainPanel.add(filePanel);

        // Create the border
        TitledBorder title =
BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED), "File Directory");
        title.setTitleJustification(TitledBorder.LEFT);
        filePanel.setBorder(title);

        // Add the file directory field
        c.gridx = 0;

```

```

        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        filePanel.add(directoryLabel, c);
        directoryField = new JTextField(32);
        c.gridx = 1;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        filePanel.add(directoryField, c);

        // Add the browse button
        browseButton = new JButton("Browse");
        browseButton.addActionListener(this);
        fileChooser = new JFileChooser();
        fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        c.gridx = 2;
        c.gridy = 0;
        c.insets = new Insets(5,6,5,5);
        filePanel.add(browseButton, c);
    }

    // Creates the stream settings section of the GUI
    private void createStreamUI() {
        GridBagLayout g = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        JPanel streamPanel = new JPanel(g);

        mainPanel.add(streamPanel);

        // Create the border
        TitledBorder title =
BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(EtchedBorder.LOWERED), "Waste Streams");
        title.setTitleJustification(TitledBorder.LEFT);
        streamPanel.setBorder(title);

        // Add the fields for stream 1
        stream1Enable = new JCheckBox();
        stream1Enable.setSelected(true);
        c.gridx = 0;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        streamPanel.add(stream1Enable, c);
        stream1Enable.addItemListener(this);
        c.gridx = 1;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        streamPanel.add(stream1Label, c);
        stream1Field = new JTextField(10);
        c.gridx = 2;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        streamPanel.add(stream1Field, c);
        c.gridx = 3;
        c.gridy = 0;
        c.insets = new Insets(5,20,5,0);
        streamPanel.add(tare1Label, c);
        tareFields[0] = new JTextField(4);
        c.gridx = 4;
        c.gridy = 0;
        c.insets = new Insets(5,5,5,0);
        streamPanel.add(tareFields[0], c);
        c.gridx = 5;

```

```

c.gridy = 0;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare1UnitsLabel, c);
tare1aEnable = new JCheckBox();
tare1aEnable.setSelected(false);
c.gridx = 6;
c.gridy = 0;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare1aEnable, c);
tare1aEnable.addItemListener(this);
c.gridx = 7;
c.gridy = 0;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare1aLabel, c);
tareFields[5] = new JTextField(4);
c.gridx = 8;
c.gridy = 0;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[5], c);
tareFields[5].setEnabled(false);
c.gridx = 9;
c.gridy = 0;
c.insets = new Insets(5,5,5,5);
streamPanel.add(tare1aUnitsLabel, c);

// Add the fields for stream 2
stream2Enable = new JCheckBox();
stream2Enable.setSelected(true);
c.gridx = 0;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream2Enable, c);
stream2Enable.addItemListener(this);
c.gridx = 1;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream2Label, c);
stream2Field = new JTextField(10);
c.gridx = 2;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream2Field, c);
c.gridx = 3;
c.gridy = 1;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare2Label, c);
tareFields[1] = new JTextField(4);
c.gridx = 4;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[1], c);
c.gridx = 5;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare2UnitsLabel, c);
tare2aEnable = new JCheckBox();
tare2aEnable.setSelected(false);
c.gridx = 6;
c.gridy = 1;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare2aEnable, c);
tare2aEnable.addItemListener(this);

```

```
c.gridx = 7;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare2aLabel, c);
tareFields[6] = new JTextField(4);
c.gridx = 8;
c.gridy = 1;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[6], c);
tareFields[6].setEnabled(false);
c.gridx = 9;
c.gridy = 1;
c.insets = new Insets(5,5,5,5);
streamPanel.add(tare2aUnitsLabel, c);
```

```
// Add the fields for stream 3
stream3Enable = new JCheckBox();
stream3Enable.setSelected(true);
c.gridx = 0;
c.gridy = 2;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream3Enable, c);
stream3Enable.addItemListener(this);
c.gridx = 1;
c.gridy = 2;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream3Label, c);
stream3Field = new JTextField(10);
c.gridx = 2;
c.gridy = 2;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream3Field, c);
c.gridx = 3;
c.gridy = 2;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare3Label, c);
tareFields[2] = new JTextField(4);
c.gridx = 4;
c.gridy = 2;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[2], c);
c.gridx = 5;
c.gridy = 2;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare3UnitsLabel, c);
tare3aEnable = new JCheckBox();
tare3aEnable.setSelected(false);
c.gridx = 6;
c.gridy = 2;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare3aEnable, c);
tare3aEnable.addItemListener(this);
c.gridx = 7;
c.gridy = 2;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare3aLabel, c);
tareFields[7] = new JTextField(4);
c.gridx = 8;
c.gridy = 2;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[7], c);
tareFields[7].setEnabled(false);
```

```
c.gridx = 9;
c.gridy = 2;
c.insets = new Insets(5,5,5,5);
streamPanel.add(tare3aUnitsLabel, c);
```

```
// Add the fields for stream 4
stream4Enable = new JCheckBox();
stream4Enable.setSelected(true);
c.gridx = 0;
c.gridy = 3;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream4Enable, c);
stream4Enable.addItemListener(this);
c.gridx = 1;
c.gridy = 3;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream4Label, c);
stream4Field = new JTextField(10);
c.gridx = 2;
c.gridy = 3;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream4Field, c);
c.gridx = 3;
c.gridy = 3;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare4Label, c);
tareFields[3] = new JTextField(4);
c.gridx = 4;
c.gridy = 3;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[3], c);
c.gridx = 5;
c.gridy = 3;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare4UnitsLabel, c);
tare4aEnable = new JCheckBox();
tare4aEnable.setSelected(false);
c.gridx = 6;
c.gridy = 3;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare4aEnable, c);
tare4aEnable.addItemListener(this);
c.gridx = 7;
c.gridy = 3;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare4aLabel, c);
tareFields[8] = new JTextField(4);
c.gridx = 8;
c.gridy = 3;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[8], c);
tareFields[8].setEnabled(false);
c.gridx = 9;
c.gridy = 3;
c.insets = new Insets(5,5,5,5);
streamPanel.add(tare4aUnitsLabel, c);
```

```
// Add the fields for stream 5
stream5Enable = new JCheckBox();
stream5Enable.setSelected(true);
c.gridx = 0;
c.gridy = 4;
```

```

c.insets = new Insets(5,5,5,0);
streamPanel.add(stream5Enable, c);
stream5Enable.addItemListener(this);
c.gridx = 1;
c.gridy = 4;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream5Label, c);
stream5Field = new JTextField(10);
c.gridx = 2;
c.gridy = 4;
c.insets = new Insets(5,5,5,0);
streamPanel.add(stream5Field, c);
c.gridx = 3;
c.gridy = 4;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare5Label, c);
tareFields[4] = new JTextField(4);
c.gridx = 4;
c.gridy = 4;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[4], c);
c.gridx = 5;
c.gridy = 4;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare5UnitsLabel, c);
tare5aEnable = new JCheckBox();
tare5aEnable.setSelected(false);
c.gridx = 6;
c.gridy = 4;
c.insets = new Insets(5,20,5,0);
streamPanel.add(tare5aEnable, c);
tare5aEnable.addItemListener(this);
c.gridx = 7;
c.gridy = 4;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tare5aLabel, c);
tareFields[9] = new JTextField(4);
c.gridx = 8;
c.gridy = 4;
c.insets = new Insets(5,5,5,0);
streamPanel.add(tareFields[9], c);
tareFields[9].setEnabled(false);
c.gridx = 9;
c.gridy = 4;
c.insets = new Insets(5,5,5,5);
streamPanel.add(tare5aUnitsLabel, c);
}

// Purpose: Creates the button of the GUI
private void createStartupUI() {
    JPanel startupPanel = new JPanel(new GridLayout(1, 4, 10, 10));

    mainPanel.add(startupPanel);

    // Create start button
    startButton = new JButton("Start Application");
    startButton.addActionListener(this);
    startupPanel.add(startButton);

    // Create start button
    offlineModeButton = new JButton("Start in Offline Mode");
    offlineModeButton.addActionListener(this);
}

```



```

startupPanel.add(offlineModeButton);

// Create save button
saveButton = new JButton("Save Settings");
saveButton.addActionListener(this);
startupPanel.add(saveButton);

// Create credits button
creditsButton = new JButton("About");
creditsButton.addActionListener(this);
startupPanel.add(creditsButton);
}

@SuppressWarnings("unchecked")
public List<String> getAvailableSerialPorts() {
    List<String> portList = new ArrayList<String>();

    // By default, contain a bunch of spaces
    portList.add(" ");

    Enumeration<CommPortIdentifier> ports = CommPortIdentifier.getPortIdentifiers();

    while(ports.hasMoreElements()) {
        CommPortIdentifier port = (CommPortIdentifier)ports.nextElement();
        if(port.getPortType() == CommPortIdentifier.PORT_SERIAL) {
            portList.add(port.getName());
        }
    }

    return portList;
}

private void setSettingsPath() {
    try {
        settingsPath = new File("").getCanonicalPath() + "\\settings.txt";
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public Connection getConnection() {
    return connection;
}

public boolean checkConnection() {
    return connected;
}

private void openSettings() {
    try {
        // Set the settings.txt file path
        setSettingsPath();

        // Open the file in write mode if it exists, otherwise create a new file
        BufferedReader fileReader = new BufferedReader(new FileReader(settingsPath));
        String currentLine;

        // Parse the file and enter fields
        while((currentLine = fileReader.readLine()) != null) {
            if(currentLine.contains("username ")) {
                usernameField.setText(currentLine.substring("username ".length(),
currentLine.length()));

```

```

        } else if(currentLine.contains("ipAddress ")) {
            ipAddressField.setText(currentLine.substring("ipAddress ".length(),
currentLine.length()));
        } else if(currentLine.contains("port ")) {
            portField.setText(currentLine.substring("port ".length(),
currentLine.length()));
        } else if(currentLine.contains("databaseName ")) {
            databaseNameField.setText(currentLine.substring("databaseName
".length(), currentLine.length()));
        } else if(currentLine.contains("directory ")) {
            directoryField.setText(currentLine.substring("directory ".length(),
currentLine.length()));
        } else if(currentLine.contains("indicator ")) {
            String indicator = currentLine.substring("indicator ".length(),
currentLine.length());

            for(int i = 0; i < indicatorField.getItemCount(); i++) {
                if(indicator.equals((String)indicatorField.getItemAt(i))) {
                    indicatorField.setSelectedIndex(i);
                }
            }
        } else if(currentLine.contains("control ")) {
            String control = currentLine.substring("control ".length(),
currentLine.length());

            for(int i = 0; i < controlField.getItemCount(); i++) {
                if(control.equals((String)controlField.getItemAt(i))) {
                    controlField.setSelectedIndex(i);
                }
            }
        } else if(currentLine.contains("stream1Enable ")) {
            stream1Enable.setSelected(Boolean.parseBoolean(currentLine.substring("stream1Enable ".length(),
currentLine.length())));
        } else if(currentLine.contains("stream2Enable ")) {
            stream2Enable.setSelected(Boolean.parseBoolean(currentLine.substring("stream2Enable ".length(),
currentLine.length())));
        } else if(currentLine.contains("stream3Enable ")) {
            stream3Enable.setSelected(Boolean.parseBoolean(currentLine.substring("stream3Enable ".length(),
currentLine.length())));
        } else if(currentLine.contains("stream4Enable ")) {
            stream4Enable.setSelected(Boolean.parseBoolean(currentLine.substring("stream4Enable ".length(),
currentLine.length())));
        } else if(currentLine.contains("stream5Enable ")) {
            stream5Enable.setSelected(Boolean.parseBoolean(currentLine.substring("stream5Enable ".length(),
currentLine.length())));
        } else if(currentLine.contains("stream1 ")) {
            stream1Field.setText(currentLine.substring("stream1 ".length(),
currentLine.length()));
        } else if(currentLine.contains("stream2 ")) {
            stream2Field.setText(currentLine.substring("stream2 ".length(),
currentLine.length()));
        } else if(currentLine.contains("stream3 ")) {
            stream3Field.setText(currentLine.substring("stream3 ".length(),
currentLine.length()));
        } else if(currentLine.contains("stream4 ")) {
            stream4Field.setText(currentLine.substring("stream4 ".length(),
currentLine.length()));
        } else if(currentLine.contains("stream5 ")) {

```

```

        stream5Field.setText(currentLine.substring("stream5 ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare1 ")) {
        tareFields[0].setText(currentLine.substring("tare1 ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare2 ")) {
        tareFields[1].setText(currentLine.substring("tare2 ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare3 ")) {
        tareFields[2].setText(currentLine.substring("tare3 ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare4 ")) {
        tareFields[3].setText(currentLine.substring("tare4 ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare5 ")) {
        tareFields[4].setText(currentLine.substring("tare5 ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare1aEnable ")) {
        tare1aEnable.setSelected(Boolean.parseBoolean(currentLine.substring("tare1aEnable ".length(),
currentLine.length())));
    } else if(currentLine.contains("tare2aEnable ")) {
        tare2aEnable.setSelected(Boolean.parseBoolean(currentLine.substring("tare2aEnable ".length(),
currentLine.length())));
    } else if(currentLine.contains("tare3aEnable ")) {
        tare3aEnable.setSelected(Boolean.parseBoolean(currentLine.substring("tare3aEnable ".length(),
currentLine.length())));
    } else if(currentLine.contains("tare4aEnable ")) {
        tare4aEnable.setSelected(Boolean.parseBoolean(currentLine.substring("tare4aEnable ".length(),
currentLine.length())));
    } else if(currentLine.contains("tare5aEnable ")) {
        tare5aEnable.setSelected(Boolean.parseBoolean(currentLine.substring("tare5aEnable ".length(),
currentLine.length())));
    } else if(currentLine.contains("tare1a ")) {
        tareFields[5].setText(currentLine.substring("tare1a ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare2a ")) {
        tareFields[6].setText(currentLine.substring("tare2a ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare3a ")) {
        tareFields[7].setText(currentLine.substring("tare3a ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare4a ")) {
        tareFields[8].setText(currentLine.substring("tare4a ".length(),
currentLine.length()));
    } else if(currentLine.contains("tare5a ")) {
        tareFields[9].setText(currentLine.substring("tare5a ".length(),
currentLine.length()));
    }
}

fileReader.close();
} catch (FileNotFoundException e) {
    System.out.println("Settings file not found");
    return;
} catch (IOException e) {
    e.printStackTrace();
}
}

```

```

}

/*
 * connects to Oracle database named ug using user supplied username and
 * password
 */
private boolean connect() {
    ipAddress = ipAddressField.getText();
    port = portField.getText();
    databaseName = databaseNameField.getText();
    username = usernameField.getText();

    String connectURL = "jdbc:sqlserver://" + ipAddress + ":" + port + ";databaseName=" +
databaseName;

    try {
        connection = DriverManager.getConnection(connectURL, username, new
String(passwordField.getPassword()));
        //connection = DriverManager.getConnection(connectURL, "Dash", "ubccapstone-47");
        return true;
    } catch (SQLException ex) {
        System.out.println("Message: " + ex.getMessage());
        return false;
    }
}

public void itemStateChanged(ItemEvent e) {
    int state = e.getStateChange();
    Object source = e.getItemSelectable();

    if(source == stream1Enable) {
        if(state == ItemEvent.DESELECTED) {
            stream1Field.setEnabled(false);
            tareFields[0].setEnabled(false);
            tareFields[5].setEnabled(false);
            tare1aEnable.setEnabled(false);
        } else if(state == ItemEvent.SELECTED) {
            stream1Field.setEnabled(true);
            tareFields[0].setEnabled(true);
            tareFields[5].setEnabled(true);
            tare1aEnable.setEnabled(true);
        }
    } else if(source == stream2Enable) {
        if(state == ItemEvent.DESELECTED) {
            stream2Field.setEnabled(false);
            tareFields[1].setEnabled(false);
            tareFields[6].setEnabled(false);
            tare2aEnable.setEnabled(false);
        } else if(state == ItemEvent.SELECTED) {
            stream2Field.setEnabled(true);
            tareFields[1].setEnabled(true);
            tareFields[6].setEnabled(true);
            tare2aEnable.setEnabled(true);
        }
    } else if(source == stream3Enable) {
        if(state == ItemEvent.DESELECTED) {
            stream3Field.setEnabled(false);
            tareFields[2].setEnabled(false);
            tareFields[7].setEnabled(false);
            tare3aEnable.setEnabled(false);
        } else if(state == ItemEvent.SELECTED) {
            stream3Field.setEnabled(true);

```

```

        tareFields[2].setEnabled(true);
        tareFields[7].setEnabled(true);
        tare3aEnable.setEnabled(true);
    }
} else if(source == stream4Enable) {
    if(state == ItemEvent.DESELECTED) {
        stream4Field.setEnabled(false);
        tareFields[3].setEnabled(false);
        tareFields[8].setEnabled(false);
        tare4aEnable.setEnabled(false);
    } else if(state == ItemEvent.SELECTED) {
        stream4Field.setEnabled(true);
        tareFields[3].setEnabled(true);
        tareFields[8].setEnabled(true);
        tare4aEnable.setEnabled(true);
    }
} else if(source == stream5Enable) {
    if(state == ItemEvent.DESELECTED) {
        stream5Field.setEnabled(false);
        tareFields[4].setEnabled(false);
        tareFields[9].setEnabled(false);
        tare5aEnable.setEnabled(false);
    } else if(state == ItemEvent.SELECTED) {
        stream5Field.setEnabled(true);
        tareFields[4].setEnabled(true);
        tareFields[9].setEnabled(true);
        tare5aEnable.setEnabled(true);
    }
} else if(source == tare1aEnable) {
    if(state == ItemEvent.DESELECTED) {
        tareFields[5].setEnabled(false);
    } else if(state == ItemEvent.SELECTED) {
        tareFields[5].setEnabled(true);
    }
} else if(source == tare2aEnable) {
    if(state == ItemEvent.DESELECTED) {
        tareFields[6].setEnabled(false);
    } else if(state == ItemEvent.SELECTED) {
        tareFields[6].setEnabled(true);
    }
} else if(source == tare3aEnable) {
    if(state == ItemEvent.DESELECTED) {
        tareFields[7].setEnabled(false);
    } else if(state == ItemEvent.SELECTED) {
        tareFields[7].setEnabled(true);
    }
} else if(source == tare4aEnable) {
    if(state == ItemEvent.DESELECTED) {
        tareFields[8].setEnabled(false);
    } else if(state == ItemEvent.SELECTED) {
        tareFields[8].setEnabled(true);
    }
} else if(source == tare5aEnable) {
    if(state == ItemEvent.DESELECTED) {
        tareFields[9].setEnabled(false);
    } else if(state == ItemEvent.SELECTED) {
        tareFields[9].setEnabled(true);
    }
}
}
}
/*

```

```

* event handler for startup UI window
*/
public void actionPerformed(ActionEvent e) {
    if(indicatorField == e.getSource()) {
        indicatorPort = (String)indicatorField.getSelectedltem();
    } else if(controlField == e.getSource()) {
        controlPort = (String)controlField.getSelectedltem();
    } else if(refreshButton == e.getSource()) {
        // Save the currently selected ports
        String indicator = (String)indicatorField.getSelectedltem();
        String control = (String)controlField.getSelectedltem();

        // Get the list of serial ports
        List<String> list = getAvailableSerialPorts();
        String[] portList = list.toArray(new String[list.size()]);

        // Remove all of the items
        indicatorField.removeAllItems();
        controlField.removeAllItems();

        // Repopulate the list
        for(int i = 0; i < portList.length; i++) {
            indicatorField.addItem((String)portList[i]);
            controlField.addItem((String)portList[i]);
        }

        // Select the first element by default
        indicatorField.setSelectedIndex(0);
        controlField.setSelectedIndex(0);

        // Iterate through and see if the old item could be found
        for(int i = 0; i < indicatorField.getItemCount(); i++) {
            if(indicator.equals((String)indicatorField.getItemAt(i))) {
                indicatorField.setSelectedIndex(i);
            }
        }
        for(int i = 0; i < controlField.getItemCount(); i++) {
            if(control.equals((String)controlField.getItemAt(i))) {
                controlField.setSelectedIndex(i);
            }
        }
    } else if(browseButton == e.getSource()) {
        int returnVal = fileChooser.showOpenDialog(mainFrame);

        if(returnVal == JFileChooser.APPROVE_OPTION) {
            fileDirectory = fileChooser.getSelectedFile().getPath();
            directoryField.setText(fileDirectory);
        }
    } else if(saveButton == e.getSource()) {
        try {
            // Set the settings.txt file path
            setSettingsPath();

            // Open the file in write mode if it exists, otherwise create a new file
            BufferedWriter fileWriter = new BufferedWriter(new FileWriter(settingsPath,
false));

            // Write the settings to the file
            fileWriter.write("username " + usernameField.getText());
            fileWriter.newLine();
            fileWriter.write("ipAddress " + ipAddressField.getText());
            fileWriter.newLine();

```

```

fileWriter.write("port " + portField.getText());
fileWriter.newLine();
fileWriter.write("databaseName " + databaseNameField.getText());
fileWriter.newLine();
fileWriter.write("directory " + directoryField.getText());
fileWriter.newLine();
fileWriter.write("indicator " + (String)indicatorField.getSelectedItem());
fileWriter.newLine();
fileWriter.write("control " + (String)controlField.getSelectedItem());
fileWriter.newLine();
fileWriter.write("stream1Enable " + stream1Enable.isSelected());
fileWriter.newLine();
fileWriter.write("stream2Enable " + stream2Enable.isSelected());
fileWriter.newLine();
fileWriter.write("stream3Enable " + stream3Enable.isSelected());
fileWriter.newLine();
fileWriter.write("stream4Enable " + stream4Enable.isSelected());
fileWriter.newLine();
fileWriter.write("stream5Enable " + stream5Enable.isSelected());
fileWriter.newLine();
fileWriter.write("stream1 " + stream1Field.getText());
fileWriter.newLine();
fileWriter.write("stream2 " + stream2Field.getText());
fileWriter.newLine();
fileWriter.write("stream3 " + stream3Field.getText());
fileWriter.newLine();
fileWriter.write("stream4 " + stream4Field.getText());
fileWriter.newLine();
fileWriter.write("stream5 " + stream5Field.getText());
fileWriter.newLine();
fileWriter.write("tare1 " + tareFields[0].getText());
fileWriter.newLine();
fileWriter.write("tare2 " + tareFields[1].getText());
fileWriter.newLine();
fileWriter.write("tare3 " + tareFields[2].getText());
fileWriter.newLine();
fileWriter.write("tare4 " + tareFields[3].getText());
fileWriter.newLine();
fileWriter.write("tare5 " + tareFields[4].getText());
fileWriter.newLine();
fileWriter.write("tare1aEnable " + tare1aEnable.isSelected());
fileWriter.newLine();
fileWriter.write("tare2aEnable " + tare2aEnable.isSelected());
fileWriter.newLine();
fileWriter.write("tare3aEnable " + tare3aEnable.isSelected());
fileWriter.newLine();
fileWriter.write("tare4aEnable " + tare4aEnable.isSelected());
fileWriter.newLine();
fileWriter.write("tare5aEnable " + tare5aEnable.isSelected());
fileWriter.newLine();
fileWriter.write("tare1a " + tareFields[5].getText());
fileWriter.newLine();
fileWriter.write("tare2a " + tareFields[6].getText());
fileWriter.newLine();
fileWriter.write("tare3a " + tareFields[7].getText());
fileWriter.newLine();
fileWriter.write("tare4a " + tareFields[8].getText());
fileWriter.newLine();
fileWriter.write("tare5a " + tareFields[9].getText());
fileWriter.newLine();

```

```
// Close the file
```

```

        fileWriter.close();

        // Notify the user
        JOptionPane.showMessageDialog(mainFrame, "Settings successfully saved.");
    } catch(IOException ex) {
        ex.printStackTrace();
    }
} else if(creditsButton == e.getSource()) {
    JOptionPane.showMessageDialog(mainFrame, credits, "About",
JOptionPane.PLAIN_MESSAGE);
} else if(startButton == e.getSource() || offlineModeButton == e.getSource()) {
    if(startButton == e.getSource()) {
        if(connect() == false) {
            loginAttempts++;

            if (loginAttempts >= 3) {
                JOptionPane.showMessageDialog(mainFrame, "Too many
failed login attempts. Application will now close.");
                connected = false;
                mainFrame.dispose();
                System.exit(-1);
            } else {
                JOptionPane.showMessageDialog(mainFrame, "Incorrect
username or password.");

                // Clear the password
                passwordField.setText("");
                connected = false;
            }
        }
        return;
    }
}

// Set the file path for the log file
logUI.setFileDirectory(directoryField.getText());

// Instantiate the RS-232 drivers
IndicatorDriver indicator = new IndicatorDriver();
ControlDriver control = new ControlDriver();

// Set up the streams and tares
control.setStreams(stream1Field.getText(), stream2Field.getText(),
stream3Field.getText(),
stream4Field.getText(), stream5Field.getText());
for(int i = 0; i < 10; i++) {
    try {
        tares[i] = Float.valueOf(tareFields[i].getText());
    } catch(NumberFormatException ex) {
        tares[i] = 0;
    }
}
control.setTares(tares, 10);
control.setEnabled(stream1Enable.isSelected(), stream2Enable.isSelected(),
stream3Enable.isSelected(),
stream4Enable.isSelected(), stream5Enable.isSelected());
control.setTareEnabled(tare1aEnable.isSelected(), tare2aEnable.isSelected(),
tare3aEnable.isSelected(),
tare4aEnable.isSelected(), tare5aEnable.isSelected());

// Attach the indicator driver to the control driver and vice-versa
control.withIndicatorPort(indicator);
indicator.withControlPort(control);

```



```

if(startButton == e.getSource()) {
    // Instantiate the waste table
    WasteTable wasteTable = new WasteTable();

    try {
        wasteTable.createWaste(getConnection());
    } catch(Exception ex) {
    }

    // Attach the server to the indicator driver
    indicator.withServer(getConnection(), wasteTable);
}

// Set the file path
indicator.setPath(directoryField.getText());

// Connect to the COM ports
if(indicator.connect(indicatorPort) == 0) {
    System.out.println("Scale successfully connected");
} else {
    JOptionPane.showMessageDialog(mainFrame, "Failed to initialize scale serial
port");

    indicator.close();
    control.close();
    return;
}
if(control.connect(controlPort) == 0) {
    // Send an initialization message
    control.sendData("Init\n");
    System.out.println("Initializing button panel...");

    // Check for a response for up to 5 seconds
    for(int i = 0; i < 10; i++) {
        if(control.getInitialized()) {
            break;
        } else {
            try {
                Thread.sleep(500);
            } catch (InterruptedException e1) {
                e1.printStackTrace();
            }
        }
    }

    if(control.getInitialized()) {
        System.out.println("Button panel successfully initialized.");
    } else {
        System.out.println("Failed to initialize button panel.");
        JOptionPane.showMessageDialog(mainFrame, "Failed to initialize
button panel");

        indicator.close();
        control.close();
        return;
    }
} else {
    JOptionPane.showMessageDialog(mainFrame, "Failed to initialize button panel
serial port");

    indicator.close();
    control.close();
    return;
}
}

```

```
// Start reading from the scale  
indicator.setUpTimer();
```

```
// Done setting up; hide the main window  
mainFrame.dispose();
```

```
// Show the log UI  
logUI.showLog();
```

```
connected = true;
```

```
}
```

```
}
```

```
}
```

Arduino Code

```
/******  
/* Button Panel code for the Arduino Uno */  
/*  
/* Author: Brandon Lowe */  
/* Student Number: 82090101 */  
/******  
  
#include <Wire.h>  
#include "Adafruit_LEDBackpack.h"  
#include "Adafruit_GFX.h"  
  
// Stream LEDs are set to pins 2-7, 2-4 = select, 5 = data, 6-7 = stream 5  
const int streamPins[6] = {2, 3, 4, 5, 6, 7};  
  
// Status LEDs are set to pins 8-11, 8 = stream selected, 9 = stable, 10 = success, 11 = error  
const int statusPins[4] = {8, 9, 10, 11};  
  
// Buttons are set to pins 12-17, 12-16 = streams, 17 = confirm  
const int buttonPins[6] = {16, 15, 14, 13, 12, 17};  
  
// 7-segment display  
Adafruit_7segment matrix = Adafruit_7segment();  
int weight = 0;  
  
// Flags  
boolean overCapacityBit = false;  
boolean stableBit = false;  
boolean positiveWeightBit = false;  
boolean validWeightBit = false;  
boolean streamBit = false;
```

```
boolean errorBit = false;
boolean readyBit = false;
boolean successBit = false;

const unsigned int debounceTime = 20; // Debouncing time period, in milliseconds
const int bufferSize = 128; // Serial port read buffer size in bytes
unsigned int buttonStates[6] = {0, 0, 0, 0, 0, 0}; // Previous states of the buttons, used for debouncing
unsigned long statusStates[4] = {0, 0, 0, 0}; // Status LED timing
char buffer[bufferSize]; // Buffer for receiving serial data

void setup() {
    // Set up the serial connection
    Serial.setTimeout(5000);
    Serial.begin(9600);

    initialize();
}

void loop() {
    int buttonPressed;
    int bytesToSend;
    int i;

    // Check the buttons
    buttonPressed = checkButtons();
    if(buttonPressed >= 0 && buttonPressed <= 5) {
        bytesToSend = buttonPressed + '0';
        Serial.write(bytesToSend);
    }

    // Check the RS232 port
    if(Serial.available()) {
```

```
// Read until we receive a new line character, the buffer is full or we time out
Serial.readBytesUntil('\n', buffer, bufferSize);

// Parse the data read
parseSerial();
}

// State machine outputs: LEDs, 7 seg, RS232
checkReady();
setStatusLEDs();

// Check temporary LED states, i.e. success/error LEDs, which only turn on for 3s
if(digitalRead(statusPins[2]) == HIGH && statusStates[2] < millis()) {
    successBit = false;
    digitalWrite(statusPins[2], LOW);
}
if(digitalRead(statusPins[3]) == HIGH && statusStates[3] < millis()) {
    errorBit = false;
    digitalWrite(statusPins[3], LOW);
}
}

// Purpose: Initializes the Arduino
void initialize() {
    int i;

    // Set stream LEDs as outputs
    for(i = 0; i < 6; i++) {
        pinMode(streamPins[i], OUTPUT);
    }

    // Set status LEDs as outputs
```

```
for(i = 0; i < 4; i++) {  
    pinMode(statusPins[i], OUTPUT);  
}
```

```
// Set push buttons as inputs
```

```
for(i = 0; i < 6; i++) {  
    pinMode(buttonPins[i], INPUT);  
}
```

```
// Reset all of the things
```

```
resetFlags();  
setStatusLEDs();  
setStreamLEDs(-1, -1);
```

```
// Set up the 7-segment display
```

```
matrix.begin(0x70);  
matrix.setBrightness(2);  
matrix.writeDigitRaw(0, 0);  
matrix.writeDigitRaw(1, 0);  
matrix.writeDigitRaw(3, 0);  
matrix.writeDigitRaw(4, 0);  
matrix.writeDisplay();
```

```
// Send initialization confirmation
```

```
Serial.write('!');  
}
```

```
// Purpose: Sets all of the flags to their default values
```

```
void resetFlags() {  
    int i;  
  
    overCapacityBit = false;
```

```

stableBit = false;
positiveWeightBit = false;
validWeightBit = false;
streamBit = false;
errorBit = false;
readyBit = false;
successBit = false;

for(i = 0; i < 6; i++) {
    buttonStates[i] = 0;
}

for(i = 0; i < 4; i++) {
    statusStates[i] = 0;
}
}

// Purpose: Updates the 7-segment display to show the weight
void updateDisplay() {
    if(overCapacityBit) {
        // Write "OCAP"
        matrix.writeDigitRaw(0, 63);
        matrix.writeDigitRaw(1, 57);
        matrix.writeDigitRaw(3, 119);
        matrix.writeDigitRaw(4, 115);
    } else {
        boolean leadingZero = true;
        int digit = weight / 1000;

        // First digit depends on the weight being positive or negative
        if(positiveWeightBit) {
            // Write first digit

```

```

if(digit == 0) {
    matrix.writeDigitRaw(0, 0);
} else {
    matrix.writeDigitNum(0, (weight / 1000), false);
    leadingZero = false;
}
} else {
    // Write negative sign
    matrix.writeDigitRaw(0, 64);
}

// Second digit
digit = weight / 100 % 10;
if(leadingZero == true && digit == 0) {
    matrix.writeDigitRaw(1, 0);
} else {
    matrix.writeDigitNum(1, (weight / 100) % 10, false);
}

// Third digit
matrix.writeDigitNum(3, (weight / 10) % 10, true);

// Fourth digit
matrix.writeDigitNum(4, weight % 10, false);
}

// Write the numbers to the display
matrix.writeDisplay();
}

// Purpose: Polls and debounces each of the six button input pins.
//     Only takes the last button press that registers, to prevent

```



```

//      the user from pressing multiple buttons at once
// Output: The button that has been pressed (0-5), -1 if no button pressed
int checkButtons() {
    int i;
    int tempButtonState;
    int result = -1;
    unsigned long timeLimit;

    // Check each of the buttons, return the last button state change detected
    for(i = 0; i < 6; i++) {
        // Read the button state
        tempButtonState = digitalRead(buttonPins[i]);

        // Check that the button state is different from the previous state
        if(tempButtonState != buttonStates[i]) {
            // Determine the time to stop debouncing
            timeLimit = millis() + debounceTime;

            // Check that the reading is stable
            while(millis() < timeLimit) {
                if(tempButtonState != digitalRead(buttonPins[i])) {
                    goto nextButton;
                }
            }

            // Button reading is stable, check if it's low to high
            if(tempButtonState == HIGH) {
                // Overwrite the result (only take the last button pressed)
                result = i;
            }

            // Record the button state change for future debounces

```

```

    buttonStates[i] = tempButtonState;
}

nextButton++;
}

return result;
}

// Purpose: Checks the various flags to determine whether the system is ready
// to take a weight measurement
void checkReady() {
    if(!overCapacityBit && positiveWeightBit && stableBit) {
        validWeightBit = true;
    } else {
        validWeightBit = false;
    }

    if(validWeightBit && streamBit) {
        readyBit = true;
    } else {
        readyBit = false;
    }
}

// Purpose: Checks the various flags to determine the states of the status LEDs
void setStatusLEDs() {
    if(streamBit) {
        digitalWrite(statusPins[0], HIGH);
    } else {
        digitalWrite(statusPins[0], LOW);
    }
}

```

```

if(readyBit) {
    digitalWrite(statusPins[1], HIGH);
} else {
    digitalWrite(statusPins[1], LOW);
}
if(successBit) {
    digitalWrite(statusPins[2], HIGH);
    digitalWrite(statusPins[3], LOW);
    statusStates[2] = millis() + 2000;
    successBit = false;
}
if(errorBit) {
    digitalWrite(statusPins[3], HIGH);
    digitalWrite(statusPins[2], LOW);
    statusStates[3] = millis() + 2000;
    errorBit = false;
}
}

// Purpose: Sets one of the stream LEDs to state 0 or 1, with 0 being green
//         and any other number being red. Also turns off all other stream
//         LEDs, so that at most one of the 5 LEDs is on at any given time
// Input: led - The corresponding LED to turn on, from 0-4. Send any other
//         value to turn all LEDs off
//         value - The value to set the LED to
void setStreamLEDs(int led, int value) {
    streamBit = true;
    switch(led) {
        case 0:
            if(value == 0) {
                digitalWrite(streamPins[0], LOW);
                digitalWrite(streamPins[1], LOW);
            }
        }
    }
}

```

```
digitalWrite(streamPins[2], LOW);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
} else {
digitalWrite(streamPins[0], HIGH);
digitalWrite(streamPins[1], LOW);
digitalWrite(streamPins[2], LOW);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
}
break;
```

case 1:

```
if(value == 0) {
digitalWrite(streamPins[0], LOW);
digitalWrite(streamPins[1], HIGH);
digitalWrite(streamPins[2], LOW);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
} else {
digitalWrite(streamPins[0], HIGH);
digitalWrite(streamPins[1], HIGH);
digitalWrite(streamPins[2], LOW);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
}
break;
```

case 2:

```
if(value == 0) {
```

```
digitalWrite(streamPins[0], LOW);
digitalWrite(streamPins[1], LOW);
digitalWrite(streamPins[2], HIGH);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
} else {
digitalWrite(streamPins[0], HIGH);
digitalWrite(streamPins[1], LOW);
digitalWrite(streamPins[2], HIGH);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
}
break;
case 3:
if(value == 0) {
digitalWrite(streamPins[0], LOW);
digitalWrite(streamPins[1], HIGH);
digitalWrite(streamPins[2], HIGH);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
} else {
digitalWrite(streamPins[0], HIGH);
digitalWrite(streamPins[1], HIGH);
digitalWrite(streamPins[2], HIGH);
digitalWrite(streamPins[3], HIGH);
digitalWrite(streamPins[4], LOW);
digitalWrite(streamPins[5], LOW);
}
break;
```

case 4:

```
if(value == 0) {  
    digitalWrite(streamPins[0], LOW);  
    digitalWrite(streamPins[1], LOW);  
    digitalWrite(streamPins[2], LOW);  
    digitalWrite(streamPins[3], LOW);  
    digitalWrite(streamPins[4], HIGH);  
    digitalWrite(streamPins[5], LOW);  
} else {  
    digitalWrite(streamPins[0], LOW);  
    digitalWrite(streamPins[1], LOW);  
    digitalWrite(streamPins[2], LOW);  
    digitalWrite(streamPins[3], LOW);  
    digitalWrite(streamPins[4], LOW);  
    digitalWrite(streamPins[5], HIGH);  
}  
break;
```

default:

```
streamBit = false;  
digitalWrite(streamPins[0], LOW);  
digitalWrite(streamPins[1], LOW);  
digitalWrite(streamPins[2], LOW);  
digitalWrite(streamPins[3], LOW);  
digitalWrite(streamPins[4], LOW);  
digitalWrite(streamPins[5], LOW);  
break;  
}  
}
```

// Purpose: Parses the data in the buffer containing data received from the

// serial port, and updates the state accordingly

// Notes: Example weight - "WS 394" (stable weight of 39.4kg)

```
//          - "WU4218" (unstable weight of 421.8kg)
//          - "WS- 8" (stable weight of -0.8kg)
//          - "WO" (over capacity, note that that is a capital 'o', not a zero)
// Example error - "E"
// Example success - "S"
// Example stream - "T01" (set stream 0 to the second bin weight)
//          - "T40" (set stream 4 to the first bin weight)
```

```
void parseSerial() {
```

```
    int i;
```

```
    int tempWeight = 0;
```

```
    // Check if the data is a weight, stream or status update
```

```
    if(buffer[0] == 'W') {
```

```
        if(buffer[1] == 'O') {
```

```
            // Weight is over capacity
```

```
            overCapacityBit = true;
```

```
            positiveWeightBit = true;
```

```
            updateDisplay();
```

```
            return;
```

```
        } else if(buffer[1] == 'U') {
```

```
            // Weight is unstable
```

```
            stableBit = false;
```

```
            overCapacityBit = false;
```

```
        } else if(buffer[1] == 'S') {
```

```
            // Weight is stable
```

```
            stableBit = true;
```

```
            overCapacityBit = false;
```

```
        }
```

```
    // Parse the weight
```

```
    if(buffer[2] == '-') {
```

```
        positiveWeightBit = false;
```

```

for(i = 3; i < 7; i++) {
    if(buffer[i] >= '0' && buffer[i] <= '9') {
        tempWeight *= 10;
        tempWeight += (buffer[i] - '0');
    }
}
weight = tempWeight;
} else {
    positiveWeightBit = true;
    for(i = 2; i < 7; i++) {
        if(buffer[i] >= '0' && buffer[i] <= '9') {
            tempWeight *= 10;
            tempWeight += (buffer[i] - '0');
        }
    }
    weight = tempWeight;
}

updateDisplay();
return;
} else if(buffer[0] == 'E') {
    // Turn on error LED
    errorBit = true;

    // Reset the state
    setStreamLEDs(-1, -1);
    streamBit = false;
    readyBit = false;
    successBit = false;
} else if(buffer[0] == 'S') {
    // Turn on success LED, turn off error LED
    successBit = true;

```



```
// Reset the state
setStreamLEDs(-1, -1);
streamBit = false;
readyBit = false;
errorBit = false;
} else if(buffer[0] == 'T') {
// Update the waste stream LEDs
setStreamLEDs(buffer[1] - '0', buffer[2] - '0');
} else if(buffer[0] == 'l' && buffer[1] == 'n' && buffer[2] == 'i' && buffer[3] == 't') {
initialize();
}
}
```

Appendix C: Testing Documents and Other Documentation

Main Application Testing Document:

Note: Many test cases will require a weight measurement. The values to be used for each of these test cases, marked with an asterisk (*) will be all applicable combinations of: Streams 1-5, Tares 1-2, Weight -10.0 (+/- 1), 0.0, 100.0 (+/- 1), over capacity. Take note of any failed tests and their input values at the end of this document.

User interface testing:

UI1 Server login

- UI1.1 User should be able to log in given the correct username and password combination
- UI1.2 User should not be able to log in given the incorrect username and password combination
 - UI1.2.1 User should receive a message alert given an incorrect username and password combination
 - UI1.2.2 User should be able to try to log in again after a failed attempt
 - UI1.2.3 Given that three failed login attempts have occurred, the application should notify the user that they have made too many failed login attempts and terminate
- UI1.3 Given a successful server connection, SQL table CapstoneWasteManagement should be created given that it does not exist
 - UI1.3.1 If the SQL table does exist, no new table should be created and the previously existing CapstoneWasteManagement table should be used

UI2 File directory

- UI2.1 User should be able to specify a file directory path for the files to be stored
 - UI2.1.1 Log and data files should be saved in the specified directory
 - UI2.1.2 If a folder does not exist, it should be created

UI3 Serial ports

- UI3.1 User should only be able to select from available serial ports
- UI3.2 User should be able to refresh the list of available serial ports
 - UI3.2.1 Given that a port was previously in the list, if the port is disabled/removed and the list is refreshed, the serial port should no longer appear

- UI3.2.2 Given that a port was not previously in the list, if the port is enabled/added and the list is refreshed, the serial port should appear on the list
- UI3.3 Given that the indicator is connected to a serial port and the serial port is selected for the indicator port, if the user attempts to connect, the user should be able to send and receive a weight ticket from the indicator by sending 0x05 in hex
- UI3.4 Given that the button panel is connected to a serial port and the serial port is selected for the button panel, if the user attempts to connect, the application should send "Init\n" to the button panel and wait for button panel driver's "initialized" flag to be set to true (by the button panel driver receiving "I" from the serial port)

UI4 Streams

- *UI4.1 If stream is enabled, then the user should be able to make measurements with that stream
 - *UI4.1.1 The user should be able to change the stream name, and the stream name should show up in any subsequent measurements taken with that stream
 - *UI4.1.2 The user should be able to change the first tare value to -10, 0, 100, 500, and any subsequent measurements with that tare should have a net weight value of gross - tare
 - *UI4.1.3 The user should be able to enable and set the second tare value to -10, 0, 100, 500, and any subsequent measurements with that tare should have a net weight value of gross - tare
 - *UI4.1.4 The user should be able to disable the second tare, and the user should not be able to select the second tare from that stream and instead receive an error
- *UI4.2 If the stream is disabled, then the user should not be able to make measurements and should receive an error
 - *UI4.2.1 The user should not be able to change any of the settings for a disabled stream

UI5 Saving and loading settings

- UI5.1 If the user presses Save Settings, the current settings should all be saved in a text file named "settings.txt" in the directory where the application is stored
 - UI5.1.1 The server username, IP address, port and waste table name should be saved, but the password should not be saved
 - UI5.1.2 The file directory should be saved
 - UI5.1.3 The serial ports should be saved
 - UI5.1.4 Streams 1-5 should have their enable status, name, tare 1 value, tare 2 enable status and tare 2 values saved
 - UI5.1.5 If a field is left blank on the application UI, the value to be saved should be an empty string ""

- UI5.2 Given that there is a "settings.txt" file in the directory where the application is stored, if the user starts the application, the file should be read and the settings saved on it should be automatically set to their corresponding fields
 - UI5.2.1 The server username, IP address, port and waste table name should be saved, but the password should not be loaded and set to their respective fields
 - UI5.2.2 The file directory should be loaded and set to their respective fields
 - UI5.2.3 The serial ports should be loaded and set to their respective fields
 - UI5.2.4 Streams 1-5 should have their enable status, name, tare 1 value, tare 2 enable status and tare 2 values loaded and set to their respective fields
 - UI5.2.5 If a field is left blank on the application UI, the field should remain empty or in its default state

UI6 Starting the application

- UI6.1 If the user selects online mode, the application should attempt to log in with the provided username and password values (refer to test UI1)
 - Given that the user has entered the correct username/password, the application should initialize the indicator and button panel (refer to tests UI3.3 and UI3.4)
- UI6.2 If the user selects offline mode, the application should not attempt to log in
 - The application should initialize the indicator and button panel (refer to tests UI3.3 and UI3.4)
- UI6.3 If the indicator fails to initialize, the application should return back and allow the user to connect again
- UI6.4 If the button panel fails to initialize, the application should return back and allow the user to connect again
- UI6.5 If the user cannot connect to the server after 10 seconds, the application should return back and allow the user to connect again
- UI6.6 If the user closes the settings window, the application should terminate

UI7 Log

- UI7.1 If the user successfully connects and initializes the application, a log window should show up
 - *UI7.1.1 Errors and successful weight measurements should be displayed on the log window
 - *UI7.1.2 Errors and successful weight measurements should be saved to a "log.txt" file in the selected file directory
 - *UI7.1.3 Successful weight measurements should be saved to a "yyyy-mm-dd.txt" file, where yyyy is the year, mm is the month (01 to 12) and dd is the day (01 to 31)
 - *UI7.1.4 Unsuccessful weight measurements should show an error on the log, but should not be saved in the "yyyy-mm-dd.txt" file

- UI7.2 If the user closes the log, the application should fully terminate
 - UI7.2.1 The server connection should be closed
 - UI7.2.2 The indicator and button panel serial ports should be closed

Button panel and indicator interfacing:

Note: The following tests assume that proper serial port connections have been established and should be conducted once in online mode (assuming a proper server connection) and once in offline mode

INT1 Button panel serial port

- INT1.1 The button panel serial port receives "0" in ASCII
 - INT1.1.1 If Stream 1 is enabled and is not currently selected, then set the current stream to Stream 1, set the current tare to Tare 1 and send "T00\n" through the serial port
 - INT1.1.2 If Stream 1 is enabled and is currently selected, the second tare is enabled and the first tare is currently selected, set the current stream to Stream 1 and the current tare to Tare 2 and send "T01\n" through the serial port
 - INT1.1.3 If Stream 1 is enabled and is currently selected, the second tare is enabled and the second tare is currently selected, set the current stream to Stream 1 and the current tare to Tare 1 and send "T00\n" through the serial port
 - INT1.1.4 If Stream 1 is enabled and is currently selected and the second tare is disabled, set the current stream to Stream 1, the current tare to Tare 1 and send "T00\n" through the serial port
 - INT1.1.5 If Stream 1 is disabled, set the current stream to null, the current tare to 0 and send "E\n" through the serial port
- INT1.2 The button panel serial port receives "1" in ASCII
 - INT1.2.1 If Stream 2 is enabled and is not currently selected, then set the current stream to Stream 2, set the current tare to Tare 1 and send "T10\n" through the serial port
 - INT1.2.2 If Stream 2 is enabled and is currently selected, the second tare is enabled and the first tare is currently selected, set the current stream to Stream 2 and the current tare to Tare 2 and send "T11\n" through the serial port
 - INT1.2.3 If Stream 2 is enabled and is currently selected, the second tare is enabled and the second tare is currently selected, set the current stream to Stream 2 and the current tare to Tare 1 and send "T10\n" through the serial port
 - INT1.2.4 If Stream 2 is enabled and is currently selected and the second tare is disabled, set the current stream to Stream 2, the current tare to Tare 1 and send "T10\n" through the serial port

- INT1.2.5 If Stream 2 is disabled, set the current stream to null, the current tare to 0 and send "E\n" through the serial port
- INT1.3 The button panel serial port receives "2" in ASCII
 - INT1.3.1 If Stream 3 is enabled and is not currently selected, then set the current stream to Stream 3, set the current tare to Tare 1 and send "T20\n" through the serial port
 - INT1.3.2 If Stream 3 is enabled and is currently selected, the second tare is enabled and the first tare is currently selected, set the current stream to Stream 3 and the current tare to Tare 2 and send "T21\n" through the serial port
 - INT1.3.3 If Stream 3 is enabled and is currently selected, the second tare is enabled and the second tare is currently selected, set the current stream to Stream 3 and the current tare to Tare 1 and send "T20\n" through the serial port
 - INT1.3.4 If Stream 3 is enabled and is currently selected and the second tare is disabled, set the current stream to Stream 3, the current tare to Tare 1 and send "T20\n" through the serial port
 - INT1.3.5 If Stream 3 is disabled, set the current stream to null, the current tare to 0 and send "E\n" through the serial port
- INT1.4 The button panel serial port receives "3" in ASCII
 - INT1.4.1 If Stream 4 is enabled and is not currently selected, then set the current stream to Stream 4, set the current tare to Tare 1 and send "T30\n" through the serial port
 - INT1.4.2 If Stream 4 is enabled and is currently selected, the second tare is enabled and the first tare is currently selected, set the current stream to Stream 4 and the current tare to Tare 2 and send "T31\n" through the serial port
 - INT1.4.3 If Stream 4 is enabled and is currently selected, the second tare is enabled and the second tare is currently selected, set the current stream to Stream 4 and the current tare to Tare 1 and send "T30\n" through the serial port
 - INT1.4.4 If Stream 4 is enabled and is currently selected and the second tare is disabled, set the current stream to Stream 4, the current tare to Tare 1 and send "T30\n" through the serial port
 - INT1.4.5 If Stream 4 is disabled, set the current stream to null, the current tare to 0 and send "E\n" through the serial port
- INT1.5 The button panel serial port receives "4" in ASCII
 - INT1.5.1 If Stream 5 is enabled and is not currently selected, then set the current stream to Stream 5, set the current tare to Tare 1 and send "T40\n" through the serial port
 - INT1.5.2 If Stream 5 is enabled and is currently selected, the second tare is enabled and the first tare is currently selected, set the current stream to Stream 5 and the current tare to Tare 2 and send "T41\n" through the serial port

- INT1.5.3 If Stream 5 is enabled and is currently selected, the second tare is enabled and the second tare is currently selected, set the current stream to Stream 5 and the current tare to Tare 1 and send "T40\n" through the serial port
- INT1.5.4 If Stream 5 is enabled and is currently selected and the second tare is disabled, set the current stream to Stream 5, the current tare to Tare 1 and send "T40\n" through the serial port
- INT1.5.5 If Stream 5 is disabled, set the current stream to null, the current tare to 0 and send "E\n" through the serial port
- INT1.6 The button panel serial port receives "5" in ASCII
 - INT1.6.1 If the currently selected stream is not null (i.e. the user has selected a stream), send 0x05 in hex through the indicator serial port to get a weight ticket and set the indicator's "confirmation" flag to true
 - INT1.6.2 If the currently selected stream is null (i.e. the user has not selected a stream), return an error and send "E\n" to the button panel serial port
- INT1.7 If the button panel serial port receives "I" then set the "initialized" flag to true
- INT1.8 If the button panel serial port receives a byte other than the ones mentioned above, the application should discard the byte read and do nothing

INT2 Indicator serial port

- INT2.1 A weight ticket request (0x05 in hex) should be sent through the indicator serial port every half second
- INT2.2 If the indicator serial port receives a message starting with "Weight ticket." then the message and the four subsequent lines of text received should be discarded, as that means the user pressed the Print button on the indicator
- INT2.3 If the indicator serial port receives "OCAP" then "WO" should be sent through the button panel indicator
- INT2.4 The indicator receives a 17 character string
 - INT2.4.1 If the weight is negative, then character 0 should be "-"
 - INT2.4.2 If the weight is not negative, then character 0 should be " "
 - INT2.4.3 Characters 1 through 7 should make up the weight as an ASCII decimal number with one decimal point of precision, and it should be right-justified (i.e. the number 10.0 should appear as " 10.0")
 - INT2.4.4 Characters 9 and 10 should be "kg"
 - INT2.4.5 Character 12 should be "G"
 - INT2.4.6 Given the weight is 0.0, characters 14-15 should be "CZ"
 - INT2.4.7 Given the weight is -10.0 (+/- 1), characters 14-15 should be "BZ"
 - INT2.4.8 Given the weight is 100.0 (+/- 1) and is stable, characters 14-15 should be " "
 - INT2.4.9 Given the weight is 100.0 (+/- 1) and the weight is unstable, characters 14-15 should be "MO"

- INT2.4.10 If the "confirmation" flag is set to true, the weight minus the tare is determined to be positive and stable, and the application is in online mode, the date and time should be taken and the stream, weight, units, date and time should all be saved in the "yyyy-mm-dd.txt" file and on the server, and the "confirmation" flag should be reset to false
- INT2.4.11 If the "confirmation" flag is set to true, the weight minus the tare is determined to be positive and stable and the application is in offline mode, the date and time should be taken and the stream, weight, units, date and time should all be saved in the "yyyy-mm-dd.txt" file, and the "confirmation" flag should be reset to false
- INT2.4.12 If the "confirmation" flag is set to true and the weight minus the tare is negative, return an error and write "E\n" to the button panel serial port, and set the flag to false
- INT2.4.13 If the "confirmation" flag is set to true and the weight is unstable, return an error and write "E\n" to the button panel serial port, and set the flag to false
- INT2.4.14 If the "confirmation" flag is set to false and the weight is set to 100.0 (+/- 1) and is stable, send "WSxxxxx" where x is the weight in ASCII, right-justified (e.g. a weight of 8.0 would be "WS008.0")
- INT2.4.15 If the "confirmation" flag is set to false and the weight is set to -10.0 (+/- 1) and is stable, send "WS-xxxx" where x is the absolute value of the weight in ASCII, right-justified (e.g. a weight of -8.0 would be "WS-08.0")
- INT2.4.16 If the "confirmation" flag is set to false and the weight is set to 100.0 (+/- 1) and is unstable, send "WUxxxxx" where x is the weight in ASCII, right-justified (e.g. a weight of 8.0 would be "WU008.0")
- INT2.4.15 If the "confirmation" flag is set to false and the weight is set to -10.0 (+/- 1) and is unstable, send "WU-xxxx" where x is the absolute value of the weight in ASCII, right-justified (e.g. a weight of -8.0 would be "WU-08.0")

Server Testing Document:

| Use Cases | Pass? |
|---|-------|
| Server logistics | |
| User should be able to create/modify login properties and credentials | ✓ |
| Server Connectivity | |
| 1. User should be able to login into the server provided a valid username/password | ✓ |
| 2. User should be able to connect to server over local area network (provided a valid login) | ✓ |
| 3. User should be able to change the IP address of the server (given a server is hosted at the ip address) | ✓ |
| 4. User should receive a response if server cannot connect | ✓ |
| 5. User should receive a response if incorrect login/password | ✓ |
| 6. User should be able to connect to server over the internet | ✓ |
| 7. User should be receive a message if connection is terminated or interrupted | ✓ |
| 8. User should be able to specify a database name | ✓ |
| 9. User should receive a response if specified database name does not exist | ✓ |
| Server Functionalities | |
| 10. User should be able to create a SQL wastestream table(given no previous table exists) | ✓ |
| 11. If a SQL wastestream table exists, a new table will not be created and the existing table should be used. | ✓ |
| 12. User should be able to input an entry into the database table (Given weight,time,date,stream) | ✓ |
| 13. User should be able to pull all data from the table | ✓ |
| 14. User should be able to Query for data from the table based on stream | ✓ |
| 15. User should be able to Query for the data from the table using dates | ✓ |
| 16. User should receive a response if no data is pulled (From stream) | ✓ |
| 17. User should receive a response if no data is pulled (from dates) | ✓ |
| 18. User should receive an exception if date is invalid when querying using dates | ✓ |
| 19. User should receive an exception if stream is invalid when querying using stream | ✓ |
| 20. User should be able to query the sum of weight given a wastestream | ✓ |
| 21. User should be able to query the sum of weight given a date or dateperiod | ✓ |

Dashboard Simulator Testing Document:

User interface testing:

UI1 Server login

- UI1.1 Application should log-in automatically – a message is displayed on console on success

UI2 Date selection

- UI2.1 User should be able to specify the start and end dates
 - UI2.1.1 Error message will appear if date is incorrect/incorrect format

UI3 Graph selection

- UI3.1 User should be able to select/deselect checkboxes
 - UI3.1.1 If none are selected, message will appear prompting user to select at least one

UI4 Generate Graph

- UI4.1 When pressed, will display appropriate graph(s)
 - UI4.1.1 Error message will appear if date is incorrect/incorrect format
 - UI4.1.2 If no checkboxes are selected, message will appear prompting user to select at least one
 - UI4.1.3 The user should not be able to press generate graph before the first graph appears
- UI4.2 Time graph should be zoomable
 - UI4.2.1 User may zoom in on desired portion of graph, and zoom out by clicking and dragging cursor to the left

Appendix D: User Manual

Overview

The digital waste management system is designed to offer a fast, simple and effective method of measuring the amount of waste produced by the new Student Union Building (SUB). The project was a response to the Alma Mater Society's (AMS) plans to build a new, more sustainable SUB as means of quantifying the amount of waste produced, ultimately to educate faculty and students and to reduce the amount of waste produced by the building. It was designed by fourth year Electrical and Computer Engineering students Brandon Lowe, Brian Yim, Kyle Lee and Lu Gan as part of the Capstone Design Course from September 2013 to April 2014. The digital waste management system consists of an Anyload FSP floor scale and Cardinal 204 Indicator, as well as a wall-mountable button panel and computer software backend. This document will provide instructions for setting up and operating the digital waste management system.

Specifications

System Requirements:

- Windows 7 64 Bit Operating System
- At least 128 MB of RAM
- Java JRE 7 or higher installed
- Internet connection
- Two RS-232 serial ports OR two USB ports AND two RS-232 to USB adapters

System Specifications and Features:

- Scale capacity of up to 500.0 kg and resolution of 0.1 kg
- Support for five different waste streams, each of which has:
 - A changeable stream name
 - Support for up to two custom tare values (the second tare value can be enabled or disabled)
 - Enable/disable
- Support for Microsoft SQL Server 2008

Installation

To remove the button panel from the wall:

1. Remove front cover; be careful not to damage the LEDs or push buttons
2. Remove the printed circuit board by unscrewing the nuts which mount it, and then remove all cables from the back of the board
3. Unscrew the two screws that mount the button panel to the wall
4. To reinstall, follow these steps in the reverse

To install the software:

1. Ensure that the computer meets the system requirements
2. Move the application folder to your preferred location
3. Start the software by double-clicking "Start.bat"

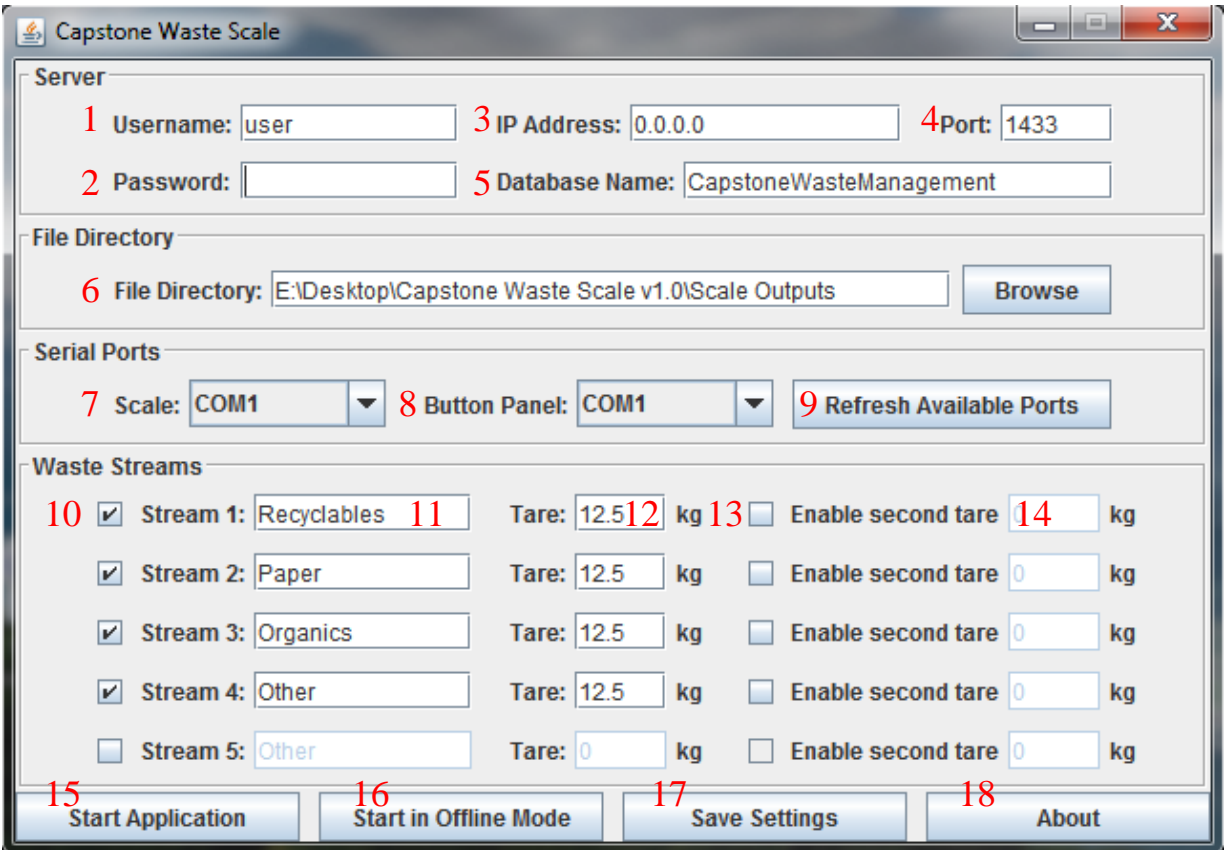
Hardware connections:

1. The scale 9-pin connector should connect to the LOAD CELL port on the indicator.
2. The SERIAL port on the indicator should connect to one of the computer's RS-232 ports with an RS-232 cable. (Note: A null modem must be connected to one side of the RS-232 cable for this connection to operate correctly)
3. The 9-pin connector on the button panel should connect to one of the computer's RS-232 ports with an RS-232 cable.
4. Both the indicator and button panel should be plugged in with their respective DC adapters. The indicator must manually be turned on, but the button panel will automatically turn on upon receiving power.

Operation

Software:

In order to start the application, open the file "Start.bat." The following window should appear.



The screenshot shows the 'Capstone Waste Scale' application window. It is divided into several sections:

- Server:** Contains fields for Username (1), Password (2), IP Address (3), and Port (4). The Database Name (5) is set to 'CapstoneWasteManagement'.
- File Directory:** Contains a File Directory field (6) with the path 'E:\Desktop\Capstone Waste Scale v1.0\Scale Outputs' and a 'Browse' button.
- Serial Ports:** Contains dropdown menus for Scale (7) and Button Panel (8), both set to 'COM1', and a 'Refresh Available Ports' button (9).
- Waste Streams:** Contains five rows, each with a checkbox (10), a Stream name field (11), a Tare field (13), and an 'Enable second tare' checkbox (14) with a kg field. The streams are: Stream 1: Recyclables (11), Tare: 12.512 kg (13), Enable second tare: 14 kg; Stream 2: Paper, Tare: 12.5 kg, Enable second tare: 0 kg; Stream 3: Organics, Tare: 12.5 kg, Enable second tare: 0 kg; Stream 4: Other, Tare: 12.5 kg, Enable second tare: 0 kg; Stream 5: Other, Tare: 0 kg, Enable second tare: 0 kg.
- Bottom Bar:** Contains four buttons: Start Application (15), Start in Offline Mode (16), Save Settings (17), and About (18).

The numbers in red show the different settings and fields:

1. Username: The server account username
2. Password: The server account password
3. IP Address: The IP address of the server
4. Port: The port on which the server is hosted on
5. Database name: The name of the SQL table to be used by the server (Note: If the table name does not exist, a new table will be created)
6. File directory: The folder in which the log and backup files will be located. Press Browse to open up a window to select a folder, or type in the directory manually
7. Scale serial port: The serial port that the indicator is connected to. (Note: You will need a null-modem on one end of the RS-232 cable connecting the indicator and the computer)
8. Button Panel serial port: The serial port that the button panel is connected to
9. Refresh: If a serial port has been added and is not in the drop-down lists, press this button to repopulate the lists

10. Waste stream enable: Check the box to enable the given stream, and uncheck it to disable the stream and any of its settings
11. Stream name: The name of the given stream
12. Stream tare: The average weight of the empty waste bin for the given stream, to be subtracted from each waste bin of that stream measured.
13. Second tare enable: Check this box to enable the second tare value for the given stream, in the event that two types of waste bins are in circulation
14. Second tare: The average weight of the second type of waste bin while empty
15. Start application: Starts the application in online mode; requires an internet connection
16. Start in offline mode: Starts the application while ignoring the server credentials and connection. Data will only be saved in the log files
17. Save settings: Saves the current settings, so that when the application is re-opened the currently selected settings will appear by default, if possible
18. About: Displays information about the application

Upon starting the application in either online or offline mode, the settings window will close and a log window will appear. The log will display the status of the program, as well as any measurements and errors that have occurred. Closing the log will terminate the program and close any connections.

Scale Operation:

To operate the scale, first ensure that the software is running and that both the indicator and button panel are powered on. To take a measurement, drag the bin onto the scale and wait 1-2 seconds until the weight settles. The current weight should display on the button panel display. Then select the correct stream by pressing one of the five push buttons on the left side of the button panel. Assuming the stream is enabled, the LED to the right of the stream should light up and the "Stream" LED should also light up. If the stream is disabled, then an error will occur and the "Error" LED will light up. If the second tare is enabled for that stream, pressing the same stream button will toggle between the two tare values, showing a green light beside the button for the first tare and a red light for the second tare.

After a stream has been selected, wait for the "Ready" LED to turn on, as that means that the weight has stabilized and an accurate reading can be taken. Once the "Ready" LED turns on, the green confirmation button at the bottom right of the panel can be pressed. If the reading is successful, then the "Success" LED will turn on and the bin can be taken off the scale, and if the reading is unsuccessful, the "Error" LED will turn on. If the "Error" LED turns on, double check that everything is set up correctly and try again.

In short:

1. Ensure everything is plugged in, turned on and the software is started
2. Put the bin on the scale
3. Select a stream
4. Wait for the "Ready" light to turn on
5. Press the green confirmation button
6. If the green "Success" light turns on, the weighing process is complete, if the red "Error" light turns on, double check everything and try again

Common causes of the "Error" light:

1. Trying to select a stream when it is disabled
2. Pressing the green confirm button without a stream selected
3. Pressing the green confirm button while the weight was unstable
4. The load was too light and came out negative after subtracting the weight of the empty bin