Objectives

The goal of this software is to provide an easy-to-use user interface that allows user to control and monitor the charging process of individual battery pack. Users are allowed to customize the amount of cells that will be charged as well as all the parameters that are needed to balance the state of charge of individual cells (e.g. voltage, current, temperature, bypass threshold). Users should be able to monitor the state of charge, voltage, current, temperature of individual cells as long as the software is running. The software must be able to run on the designated laptop in the charging station. All real-time data during the charging process needs to be backed up. The software should also be able to generate characteristic graphs out of the charging data.

Required Hardware

- I2C to USB adaptor
- Arduino Leonardo Board

Software Tools

- Java
  - Core Application
    - RXTX Library
  - Front-end GUI
    - Swing GUI library
    - AWT GUI Library
  - Database backup
- XML
  - Cell Characteristic Files

Communication Protocols

- I2C
  - Data coming from Battery Management System boards
- USB
  - Communicate to Arduino Leonardo Board
Pit Station Software Top Level Design

Pit Station Software
Top Level Block Diagram
**Detailed Descriptions:**

**I/O Section:**

*SerialReader.java:*

- Will Continuously Read Data from one of the USB Communication Port.
- This thread will report the serial data it received to the data collector thread.
- Will place the data read in a first-in-first-out buffer if the data collector thread is not listening to the data (this shouldn’t happen).

*SerialWriter.java*

- Provide an easy to use interface to send serial data with specified parameters of the serial packet.

*StreamData.java*

- Allow the MainController Thread to directly log data in an external storage device (SD card).

**Core Application:**

*MainController.java:*

- This is the main thread that is created when the program is first executed.
- This controller thread maintains the current state of the system.
- The controller is in charge of keeping the current data up to date, backup data periodically.
- The MainController can directly communicate with GUI.

*DataCollector.java*

- This thread will continuously collect data from the BMS boards by going through the following steps:
  - Send Commands through the serial port to query the voltage of each cell
  - Send Commands through the serial port to query the temperature of each cell
  - Send Commands through the serial port to query the Coulomb counting of each cell
  - Send one command to query the current value
  - Store the values fetched by the Serial Readers
  - Update the current value held by the main controller with the new values
- This thread reports to the MainControllerThread to update the information about each cell.
- This thread also asks the GUIUpdaterThread to update the GUI with the real time data.

*GUIUpdater.java*

- This is also a separate thread that needs to running along with all the other concurrent threads mentioned before.
- The sole purpose of this thread is to update the GUI with the new data acquired by the data collector thread.
**XMLParser.java**

This parser needs to be able to parse all XML data files that provide information about the cells during charging (logged by this software) and discharging (logged by SCADA).

**CellCharGenerator.java**

This class needs to be able to generate a characteristic graph based on the parsed real-time data.

**FrontEnd API**

This API provides a way for the Main Controller to access information that the GUI holds. It also provides a convenient interface to update the GUI.