Gate 3 - Design Document Approval and Gate Participants

**Project: *Bluetooth –* *Android Intern Project***

Date(s): *Fall 2013*

Project Manager: *Blaine Tiernan*

Prepared by: *Lee Easton*

Document status: \_\_ Draft \_X\_ Proposed \_\_ Validated \_\_ Approved

**Change Log**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Changes Description** | **Author** | **Date** |
| 1 | First draft | L Easton | 2013,June 10 |
| 2 | Second Draft | L Easton | 2013, July 22 |
| 3 | New Version Draft | L Easton | 2013,Oct 21 |
| 4 | Revised Draft | L Easton | 2013,Dec 3 |

Contents

[NOTE 2](#_Toc362346500)

[**1. Introduction 3**](#_Toc362346501)

[1.1 Intended Audience 3](#_Toc362346502)

[1.2 References 3](#_Toc362346503)

[1.3 Acronyms and Abbreviations 3](#_Toc362346504)

[1.4 Assumptions 4](#_Toc362346505)

[**2. System Design 5**](#_Toc362346506)

[2.1 System Overview 5](#_Toc362346507)

[2.1.1 Development of MCCU Android Application (*MRS1*) 5](#_Toc362346508)

[2.1.2 Development of Adaptation layer 5](#_Toc362346509)

[2.1.3 Design for application page development for test automation 5](#_Toc362346510)

[2.1.4 Totalflow.exe (Flash) 5](#_Toc362346511)

[2.2 Deployment Diagram 6](#_Toc362346512)

[2.3 Solution Architecture (TRS1, TRS2) 6](#_Toc362346513)

[2.4 UI Information Model (TRS4, TRS5) 8](#_Toc362346514)

[2.5 UI Screens Summary 13](#_Toc362346515)

[2.5.1 Phase 1 Screens 13](#_Toc362346516)

[**3. Totalflow Android Application Design 14**](#_Toc362346517)

[3.1 Application Design Overview 14](#_Toc362346518)

[3.2 Android Activity Lifecycle 15](#_Toc362346519)

[3.4 Totalflow Android Application Modules 17](#_Toc362346520)

[3.4.1 Session Manager 18](#_Toc362346521)

[3.4.2 File Manager 20](#_Toc362346522)

[3.5 MCCU Bluetooth Lifecycle 26](#_Toc362346523)

[3.6 MCCU IP Lifecycle 32](#_Toc362346524)

[3.7 Design Aspects for Application Performance 38](#_Toc362346525)

[3.7.1 Page Size and Corresponding Graphics 38](#_Toc362346526)

[3.7.2 Number of Threads 39](#_Toc362346527)

[3.7.3 Memory Usage 40](#_Toc362346528)

[3.7.4 Recovery from crash/hang 41](#_Toc362346529)

[3.7.6 Indicative/Responsive UI 41](#_Toc362346530)

[3.8 Register Request Protocol (TRS4) 42](#_Toc362346531)

[3.9 Register Grouping 44](#_Toc362346532)

[3.9.1 Tube/Tank Data Selection 45](#_Toc362346533)

[3.9.2 General Data 47](#_Toc362346534)

[3.9.3 Active Alarms 50](#_Toc362346535)

[3.10 Error Codes 54](#_Toc362346538)

[3.11 Help 55](#_Toc362346539)

[3.12 User Editable Fields 56](#_Toc362346540)

[3.12.1 View Options 56](#_Toc362346541)

[3.14 Formatting (TRS5) 57](#_Toc362346542)

[3.14.1 Fonts 58](#_Toc362346543)

[3.14.2 Error Scenario Handling 58](#_Toc362346544)

[3.14.3 Res Folder 58](#_Toc362346545)

[3.14.4 Action Bar Overflow 58](#_Toc362346546)

[3.14.5 Tab Navigation 58](#_Toc362346547)

[3.14.6 Expandable List View 59](#_Toc362346548)

[3.14.7 Graphs 59](#_Toc362346549)

NOTE

The Document Author has to remember the Gate 3 is the Agreement on Product Design. This is the most important milestone, as once this gate is passed the project moves out of into coding and testing phases. A comprehensive document must be produced, outlining the technical design.

1. Introduction

This document provides details over the technical architecture and design for implementing the ABB Totalflow Android application, MCCU. It baselines the sub-systems and implementation approach for the UI and the middleware layers which is followed during the application development.

1.1 Intended Audience

This document is targeted at team management, application developers, and testers involved in implementing the MCCU application.

1.2 References

1. ABB\_NGHLA\_Layout\_Specs\_220812.pdf
2. MRS – VSOMM with Blaine.docx
3. TRS.doc
4. <http://developer.android.com/guide/components/index.html>

1.3 Acronyms and Abbreviations

| **Term** | **Definition** |
| --- | --- |
| AGA-3 | American Gas Association Report No. 3, Orifice Metering of Natural Gas. Method for calculating gas volume across an Orifice Plate. This method requires low pressure readings, Differential Pressure (DP) and Static Pressure (SP). |
| AGA-7 | American Gas Association Report No. 7, Measurement of Gas by Turbine Meters. Method for calculating gas volume using a Pulse Meter. This method requires one pressure reading, Static Pressure (SP). |
| AJAX | Asynchronous JavaScript And XML |
| API | Application Programming Interface |
| CSS | Cascading Style Sheet |
| DES | Data Encryption Standard |
| HTML | Hyper Text Markup Language |
| HTTP | Hyper Text Transfer Protocol |
| HTTPS | Hyper Text Transfer Protocol Secure |
| IP | Internet Protocol |
| JS | Java Script |
| LAN | Local Area Network |
| MCCU | Mobile Collection Unit |
| MCCU | Mobile Collection Calibration Unit |
| MRS | Marketing Requirements Specification |
| RBAC | Role Based Access Control |
| SSL | Secure Socket Layer |
| SPP | Serial Port Protocol |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| TRS | Technical Requirements Specification |
| UI | User Interface |
| UUID | Universally Unique Identifier |

1.4 Assumptions

1. Target screens for the current scope have been based on the marketing requirement document (MRS – VSOM with Blaine.docx). Any further screen modifications, addition, and deletion will need fresh analysis and review from both effort and schedule perspective.
2. All the Remote Access development and testing will consider the Android devices (and their versions) as documented in the requirement document.
3. It is assumed that no language other than English is supported for the Android application.
4. All button functionality from the Android device will perform consistently throughout the application.
5. The MCCU application will have method for exporting data to a file that can be shared externally.
6. MCCU app is only going to read the following tubes in the G4:

* AGA3
* AGA7
* AGA3SU
* AGA7SU
* APILIQUIDSU
* VCONE
* VCONESU
* CORIOLUSSU
* NOZZLE
* NOZZLESU

1. Backwards compatible for old files will be available in new app versions.
2. Devices that have previously been connected to will be saved.
3. MCCU shall be able to write back to registers on G4 Device.
4. Shall have help page that is keyword searchable and explains app functionality.
5. MCCU has read and write ability for additional Automation apps:

* Gas Lift Application
* Pulse Accumulator Application (flow rate, today vol., y’day vol.)

1. MCCU has read ability for other apps:

* IO Simulation

2. System Design

2.1 System Overview

This section captures the overall ABB Totalflow Android application context and associated entities based on the marketing requirement specifications listed in the document *MRS – VSOM with Blaine.docx*. ABB Android application is developed to enable users to browse, monitor, calibrate, and share various trends and statistical values gathered by the Totalflow device.

Scope of Remote Access MCCU development work includes:

1. Development of MCCU application.
2. Development of Adaptation layer in which the application uses Bluetooth to communicate with G4 device.
3. Development of Adaptation layer in which the application uses TCP/IP to communicate with G4 device.
4. Development of Application pages for test automation.

Key Activities involved in the Remote Access MCCU implementation are described in subsequent sub sections.

2.1.1 Development of MCCU Android Application

Java application code is running on the Android OS which sends data request to the G4 device over Bluetooth and Network connection. The wireless connection runs as a separate thread(s) from the Android device to the G4 device containing existing registers and the trends that are hosted on the flash file system. The existing RBAC is not used for this communication.

Remote Access functionality of MCCU is developed as set of login pages. These login pages will be deployed on the Android device and will consist of Java language, XML, Bluetooth SPP, and IP.

2.1.2 Development of Adaptation layer

The Adaptation layer tailors the use of Java Sockets for Bluetooth and Transmission Control Protocol (TCP/IP). As long as the Totalflow devices are running DB2 register protocol, the Adaptation layer should not be impacted.

2.1.3 Design for application page development for test automation

Application page development to support multiple Android OS versions to successfully run on various supported Android devices. Look & Feel (minor differences might be there) such as UI elements, layouts, and functionality of the page should be consistent across all the Android devices so that no functionality gets blocked or the layout gets disturbed or look misaligned.

2.1.4 Totalflow.exe (Flash)

* The flash code is enhanced to support the Application Register and File Access methods. These functions include the getting and storing of Application registers.
* Provide support for the user creation/deletion/password changing, these are used to access Remote pages.
* Changes related to implementing the retrieval of the Alarm log, Current alarms and unacknowledged Alarms.
* Changes related to implementing the retrieval of the Data units.

## 2.2 Deployment Diagram









## 2.3 Solution Architecture (TRS3)

Totalflow MCCU solution is developed as an Android application that is deployed onto the Project Wiki (https://code.google.com/p/abb-interns-2013/wiki/mainpage?tm=6). This application is accessible to end users from the Android device. The access to the G4 device data is expected to be from physically co-located Android device; however, the data can be saved and accessed again for viewing at a later time.

Remote Access enables users to monitor, change, and store configuration and statistical data from the Totalflow device.

MCCU Remote Access interfaces with other applications in Totalflow G4 device to get and update the content and data associated with all the data screens.

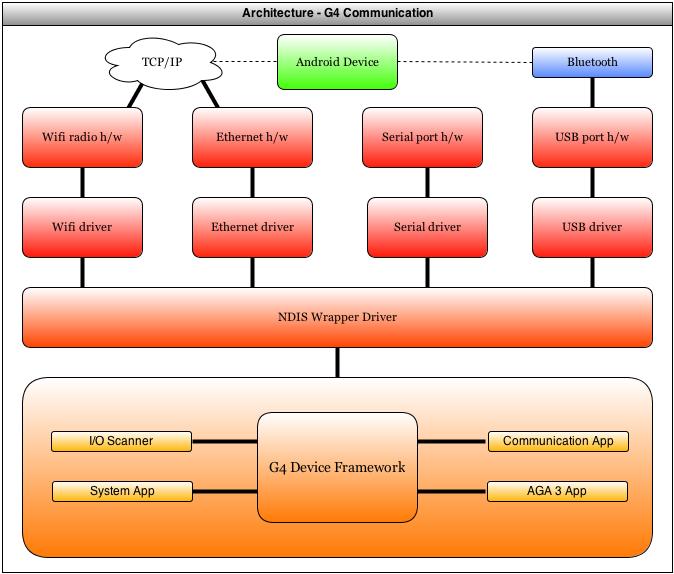


Figure 1 Solution Architecture – G4 Device

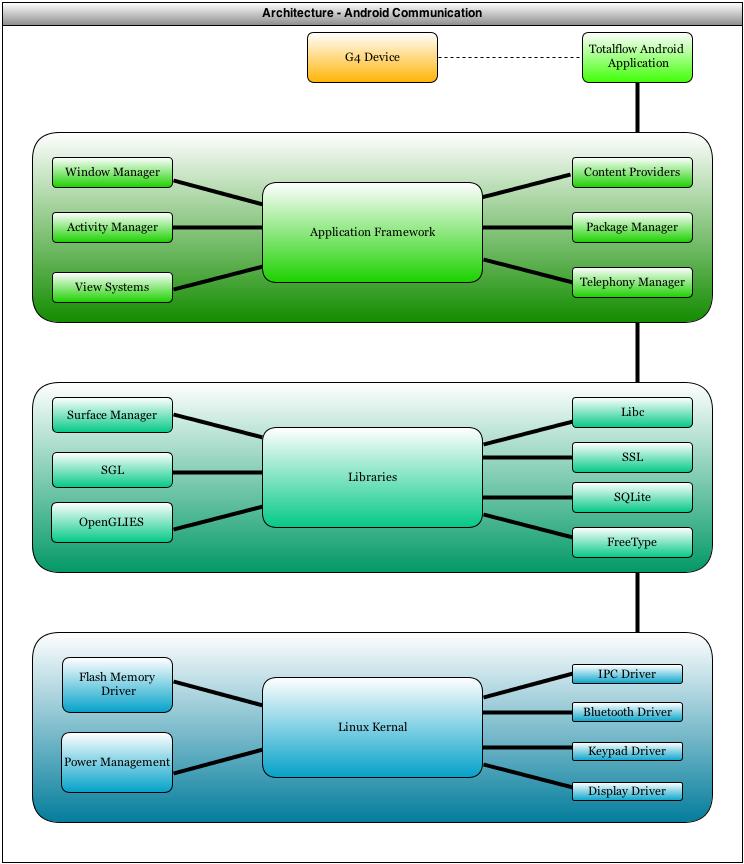


Figure 2 Solution Architecture – Android Device

## 2.4 UI Information Model (TRS4, TRS5, TRS6)

The MCCU app UI can be divided into different sets of pages:

* Home
* Wi-Fi Connection
* Bluetooth Connection
* File Explorer
* Data View
* Help
* Settings

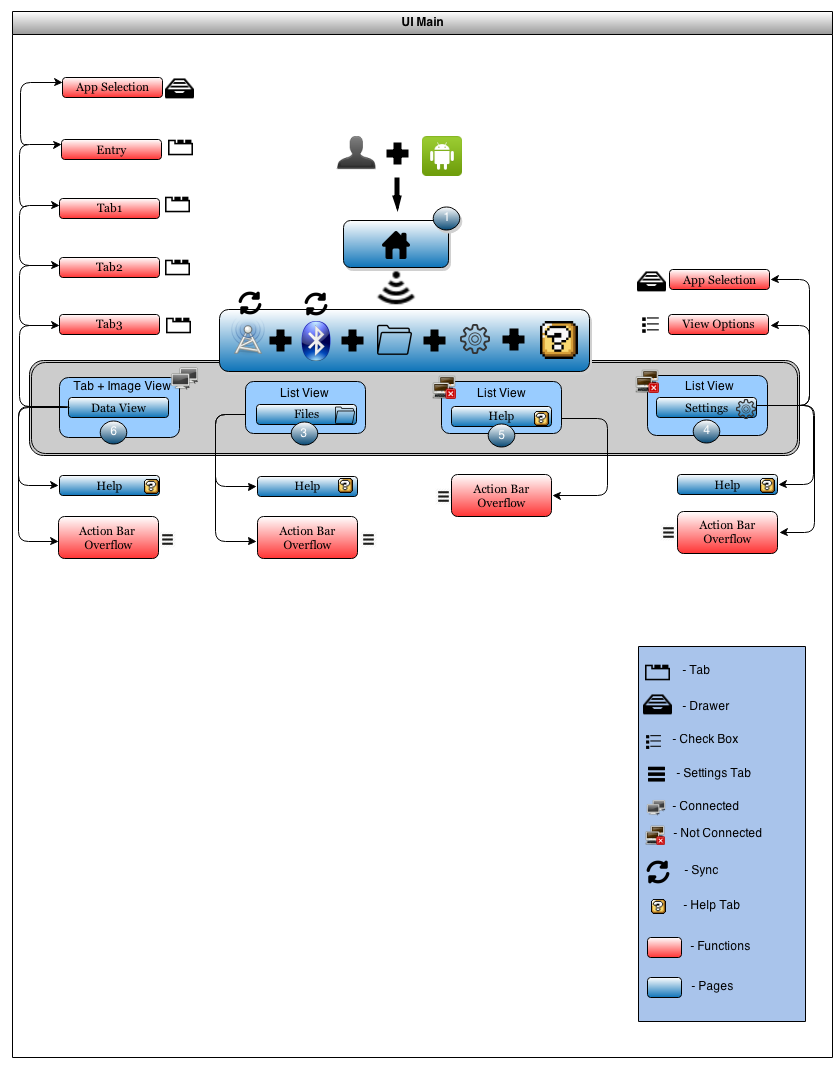


Figure 3 Login, Home Screen

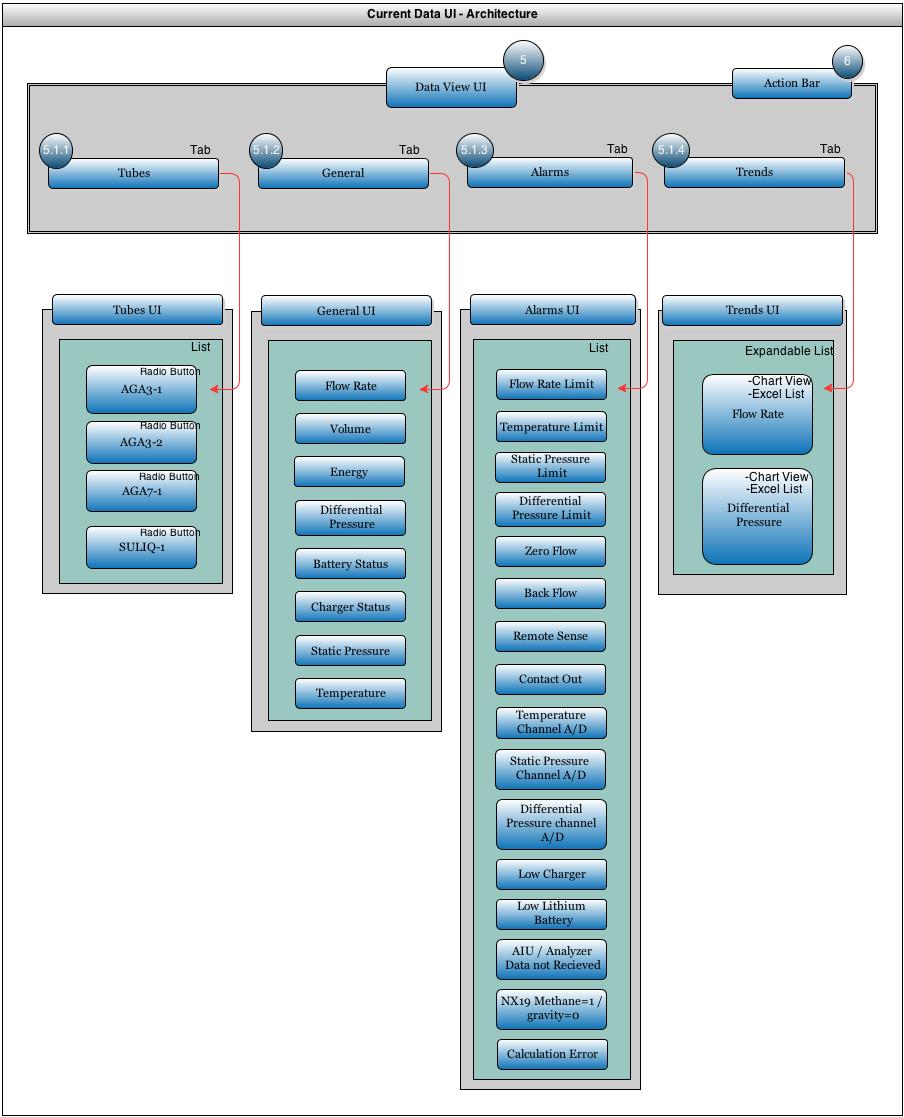


Figure 4 Data View UI

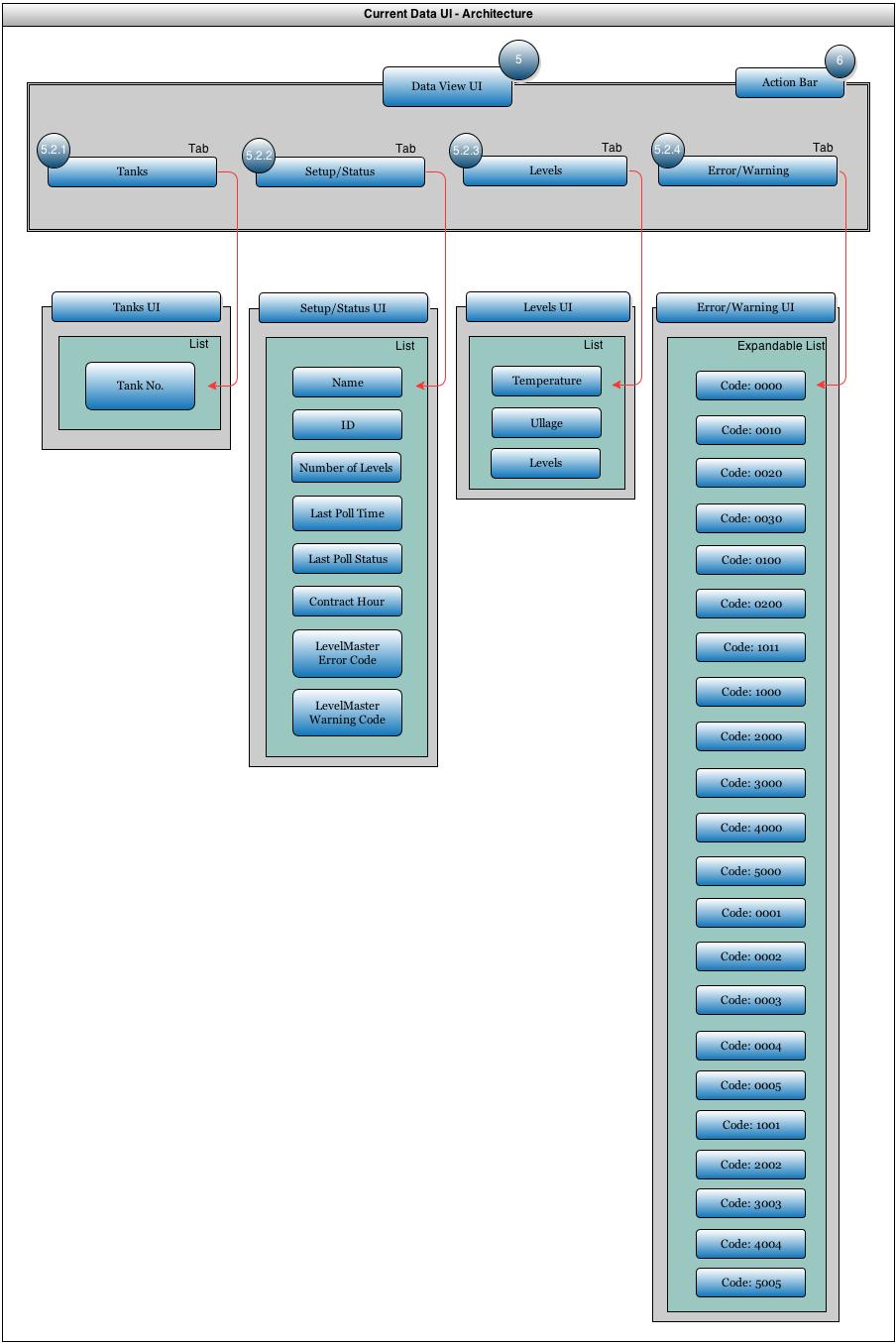


Figure 5 Data View UI (2)

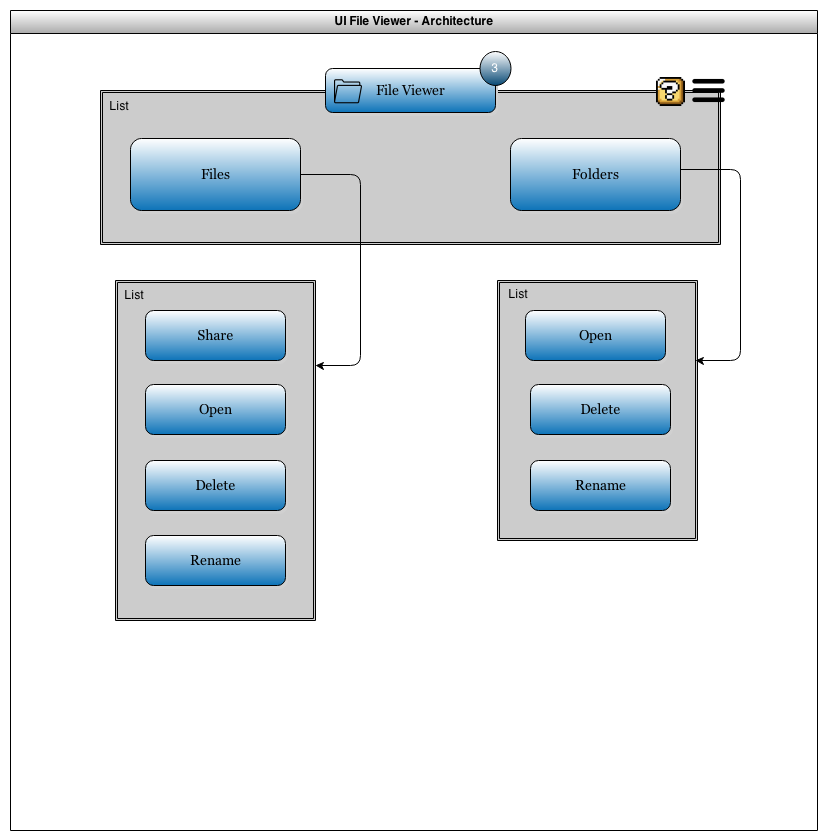


Figure 6 File View UI

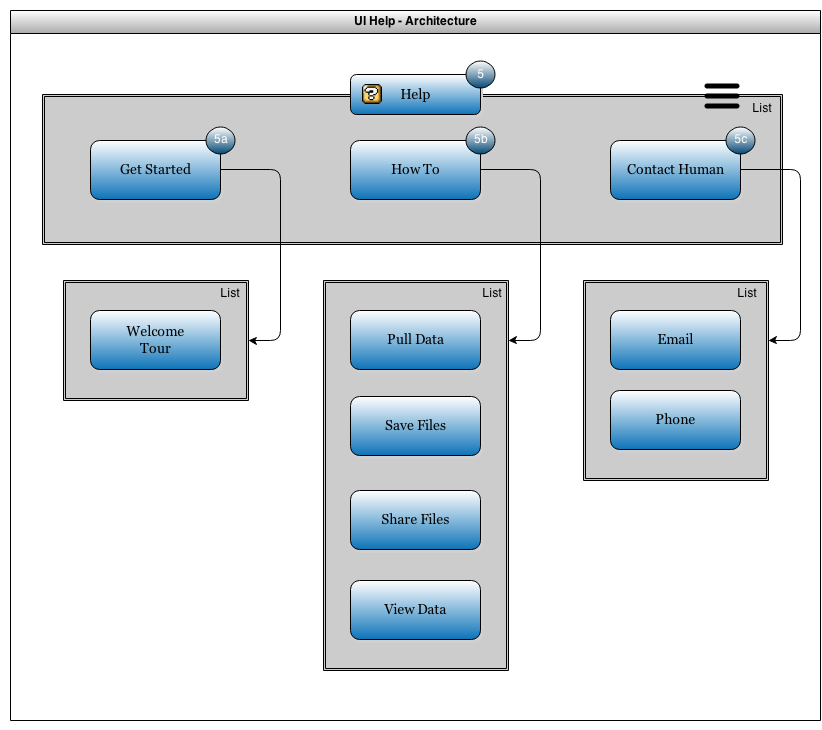


Figure 6 Help UI

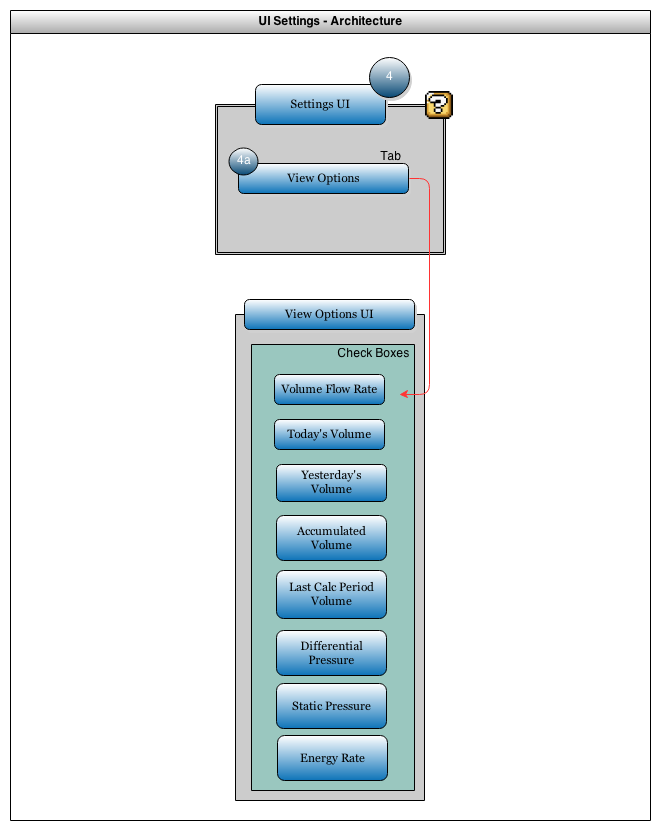


Figure 6 Settings UI

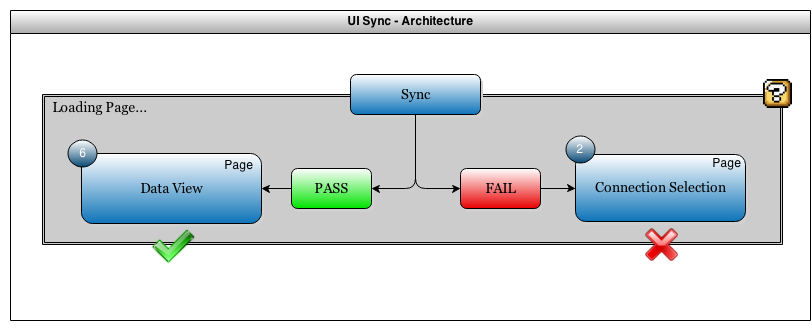


Figure 6 Sync UI

## 2.5 UI Screens Summary

## 2.5.1 Phase 1 Screens

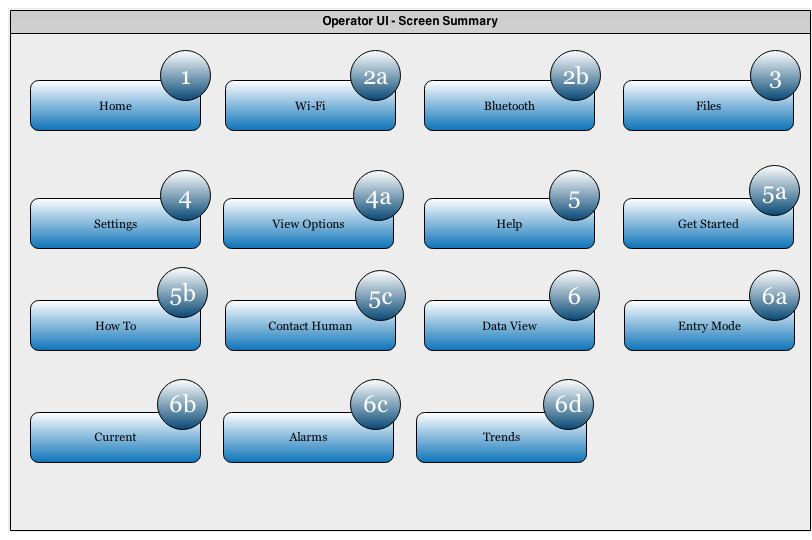


Figure 9 UI Pages Screen Summary

3. Totalflow Android Application Design

3.1 Application Design Overview

MCCU is an application running under the context of Totalflow process. This application is responsible for reading from and writing to different registers that contain current alarms and current/historical flow measurement (*TRS3*). The data is then stored on the Android’s external memory for accessing at a later time; these files shall be backward compatible to all versions of the MCCU application (*TRS1*). The MCCU application connects to the G4 devices via Bluetooth Serial Port Protocol as well as the option of TCP/IP if the G4 device sits on a network connection. After making a Bluetooth or IP connection, all settings from that connection will be saved for that specific device (*TRS2*). To help the user run the MCCU application, there shall be a task-oriented help page that provides keyword searchable results (*TRS4, TRS5*).

Scope of Design development work includes:

1. Basic Android Application Activity Lifecycle
2. Basic Android Application Indicators
3. Development of MCCU Modules.
4. Development of Class structure for Bluetooth to communicate with G4 device.
5. Development of Class structure for TCP/IP to communicate with G4 device.
6. Design for the MCCU App Array Register Access methods (Alarms and Applications).
7. Design for the MCCU File Access methods (Internal files and folders).
8. Design for the Application Page Templates.

## 3.2 Android Activity Lifecycle

The Android activity lifecycle allows for activity functionality throughout the various functions within pages of the applications.

* **Activity Launched** calls creating and starting the activity. At this point the user has selected which function they wish to use on the Application.
* **Activity Running** used after the user has selected the activity. The activity is running in the foreground and may be paused, then sent back to the start by the user if he/she shall decide to change pages and return to the activity running. There are multiple redundant paths for the user to stop the application and restart the activity. If the user decides to change or stop their activity, then the activity will be routed to **App Process Killed** and the user will then be able to create a new process.
* **App Process Killed** allows the user to begin a new activity or start the same activity from the beginning. There will only be one process running at a time, when it is killed; it is then removed from the phone’s volatile memory.
* **Activity Shut Down** is reached after the process has stopped and the user decides to close the Android application. The MCCU application only has one process running at a time, so any time they navigate pages in the app it is never destroyed and shutdown until they completely close out of the application.



Figure 10 MCCU Application Lifecycle

**3.3 Android Activity Indicators**

In the case that the Android device will reset the Bluetooth or Wi-Fi status from off/on, the indicator is shown to the user that reflects the ongoing processing by the Android application in the backend. Once the connection setting is turned on, and then the status will be displayed at the top bar of the screen

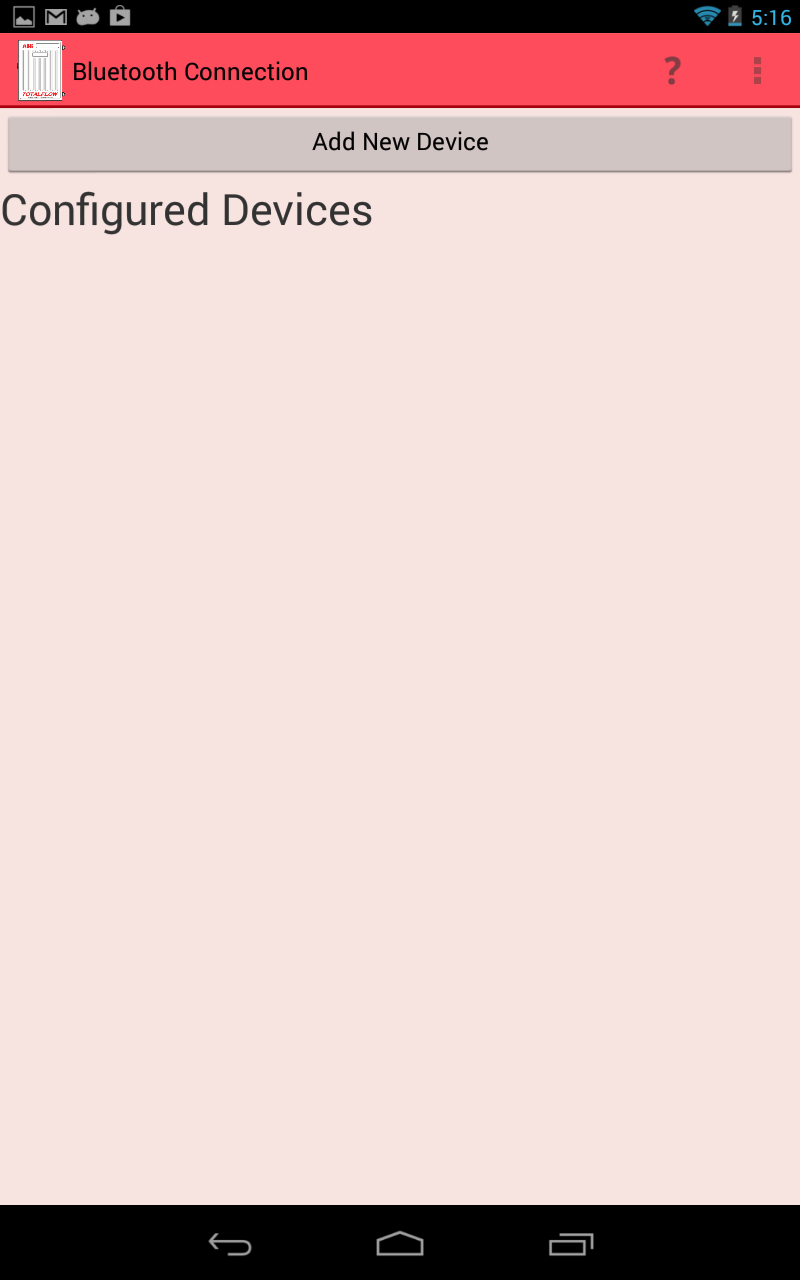
****

Figure 11 Screenshot with Bluetooth and Wi-Fi status at top right

## 3.4 MCCU Application Modules

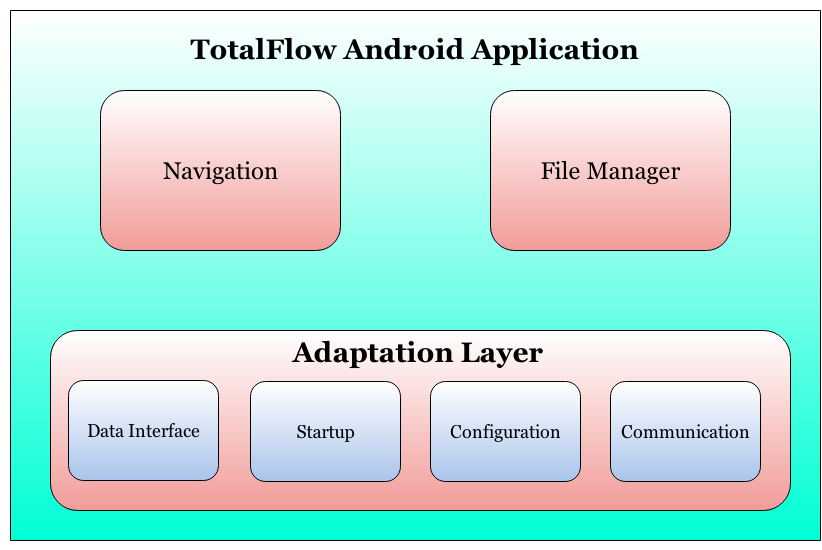


Figure 12 MCCU Application Modules

Following are the modules in MCCU Application

* Navigation
  + The use of the Action Bar will help guide navigation throughout the application.
* File Manager
  + File manager is responsible for providing user access to the Android device’s external storage.
* Adaptation Layer

This communication layer acts as a broker between the user interface and Android application framework and vice-versa.

* Starting the Remote connection
* Sending data requests
* Receiving data
* Ending the Remote connection

## 3.4.1 Navigation

Upon establishing connection to a G4 device (Application Page 2a or 2b), MCCU is configured to maintain up to one user session. The user session is limited to the duration that the user chooses to stay on the Data View page. Navigating to a different page or exiting the application will drop the connection with the G4 device. The user will be required to enter their username and password to connect to the device and navigate through the data pages. The login screens for Bluetooth and TCP/IP are mentioned in sections 3.5 and 3.6.

The MCCU navigation makes use of the libraries set by Android SDK. The Android devices’ memory will keep a list that tracks the pages that have been accessed by the user. These pages are being cached within the Android device and accessed upon request when the user presses the back button. When the user inputs their name and password, the device retrieves the register data, and then the user goes to the page to view the data. At this point the navigator tracks the page previously visited. While the user is navigating through the application pages, the Action Bar Overflow menu and the data views are kept static within the Navigator. Any other navigation through the application is a linear process (section 3.7.3).

When a new page is accessed, a new entry is made in this list. If a request comes for login at the time when there are already one active user, in that case request is refused and the user is shown an error message and taken back to page-1.

The **Action Bar** is another tool granting the user the maximum ease for application navigation. Any possible features that may need to be accessed at any point in the user session can be found in the Action Bar. The Action Bar Overflow is located in the upper right corner of the screen for new Android devices. The Action Bar Overflow is a physical touch button located below the screen for older Android devices. The Action Bar Overflow is represented by three dots inside of a square. The ease of use adds to the applications efficiency.

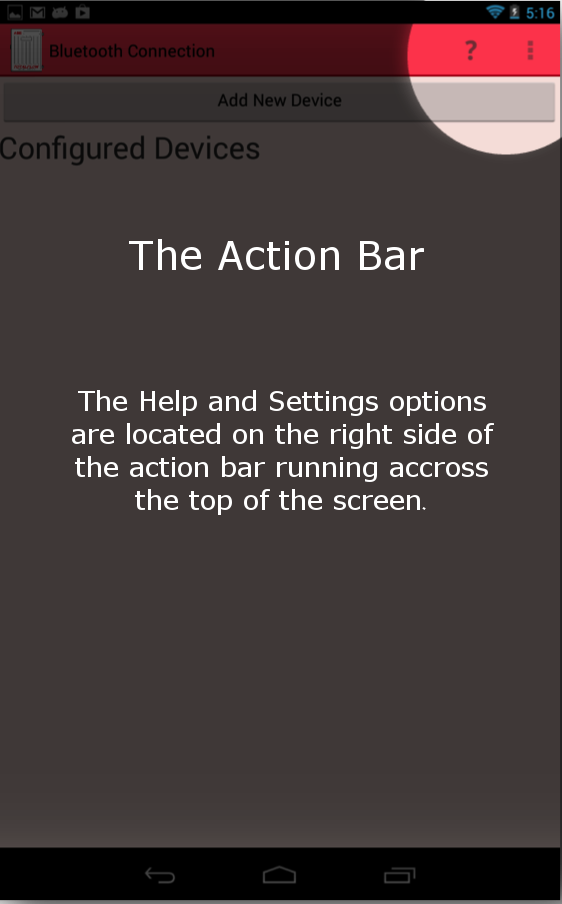
****

Figure 13 Screenshot of upper right corner where the Action Bar Overflow Tab is located

*Action Bar Overflow used throughout all of the classes:*

**// when user clicks Action Bar Overflow (Menu) Icon show the navigation menu**

*@Override*

*public boolean onCreateOptionsMenu(Menu menu) {*

**// TODO Auto-generated method stub**

*MenuInflater inflater = getMenuInflater();*

*inflater.inflate(R.menu.navigation, menu);*

*return true;*

*}*

For all items found in the Action bar such as Help, Settings, Refresh, etc. There are various functions used within the MenuInflater:

public boolean onOptionsItemSelected(MenuItem item) {

// Handle presses on the action bar items

if (mDrawerToggle.onOptionsItemSelected(item)) {

return true;

}

switch (item.getItemId()) {

case R.id.action\_help:

Intent intent = new Intent(AppContext, Help.class);

startActivity(intent);

//openSearch();

return true;

case R.id.action\_settings:

//openSettings();

return true;

case R.id.action\_save:

Intent i = new Intent(AppContext,SaveFile.class);

Bundle bundle = new Bundle();

bundle.putSerializable(appBun, applicationArrayList);

i.putExtras(bundle);

startActivity(i);

return true;

default:

//return super.onOptionsItemSelected(item);

return false;

}

}

## 3.4.2 File Manager

MCCU has the option of saving and sharing files when the user does a long-click on a file name. The user is prompted by a pop-up message giving them the option to share, delete, view, or rename.

*Creating buttons in the* ***FileViewer.java*** *class for the different pop-up actions the user can take:*

Button bOpen = (Button) optionsDialog.findViewById(R.id.bOptionOpen);

Button bDelete = (Button) optionsDialog.findViewById(R.id.bOptionDelete);

Button bRename = (Button) optionsDialog.findViewById(R.id.bOptionRename);

Button bShare = (Button) optionsDialog.findViewById(R.id.bOptionShare);

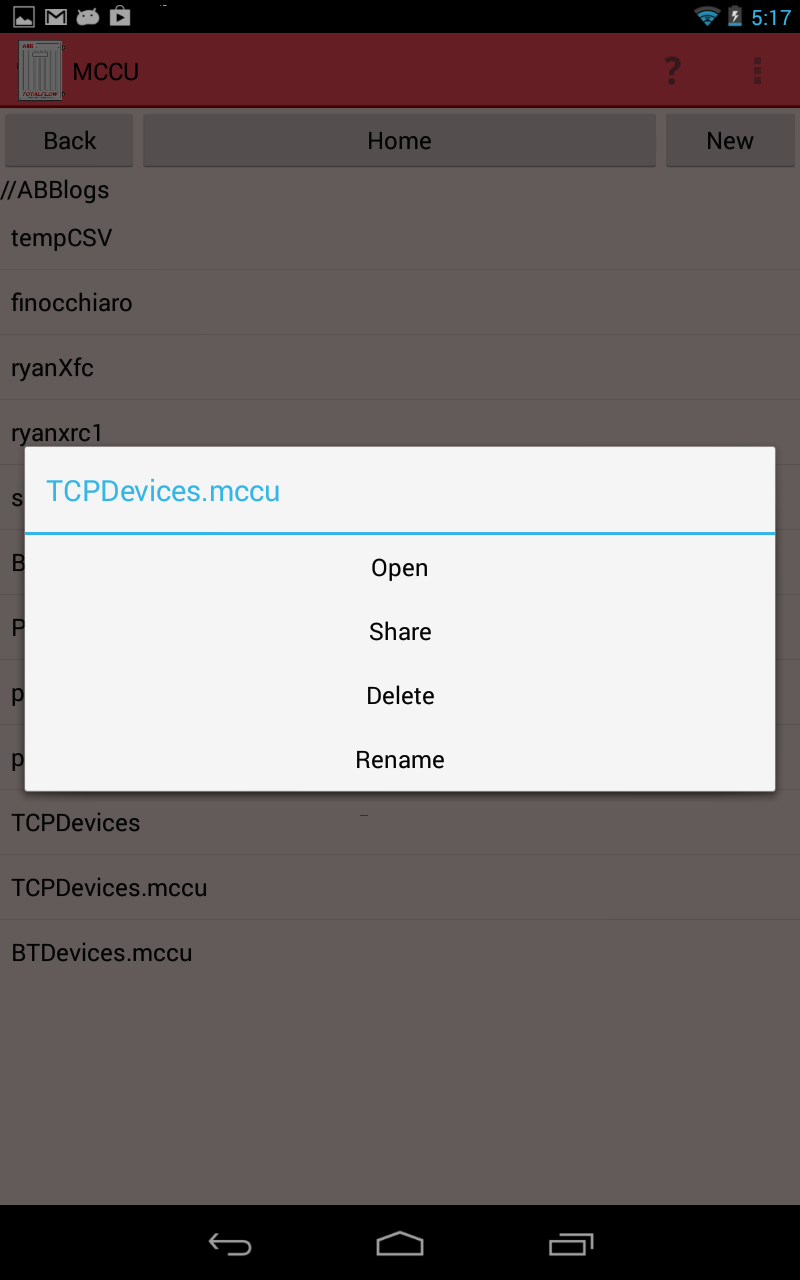
****

Figure 14 Pop-up options for managing files

**Open**

MCCU takes a file name from a list of files and launches to Appview so that the data can be displayed. If the File Explorer is on a list of folders, the view will take that folder name and re-seed it to File Explorer to display its files.

*Setting up the view from file list or folder list:*

bOpen.setOnClickListener( new OnClickListener() {

public void onClick(View arg0) {

if(absolutePathChild.isDirectory()) {

setFileList(absolutePathChild.getPath());

} else {

try {

Intent i=new Intent(context,Data\_View.class);

ObjectInputStream OBJ = new ObjectInputStream(new FileInputStream(absolutePathChild.getPath()));

Bundle bundle = new Bundle();

ArrayList<Application> apps = (ArrayList<Application>)OBJ.readObject();

bundle.putSerializable(Data\_View.appBun,apps);

bundle.putBoolean(Data\_View.bunKey,true);

i.putExtras(bundle);

startActivity(i);

}

catch (Exception e){

Log.e("MCCU", e.toString());

finish();

System.exit(1);

}

}

optionsDialog.dismiss();

}

});

**Share**

MCCU creates a set of .CSV files to share from the Android device based off of the data within the clicked file. A pop-up selection box then gives the user different options to share the .CSV file based off of the applications the user has installed on their Android device. The file is then temporarily stored on the Android’s external storage. Once the user either sends or cancels the “share” option, the file is then deleted from external storage.

*Creating the file, in the* ***Application.java*** *class, to be shared:*

public String createCsvContent(String stationID, String time) {

String csvContent=stationID+","+time+"\n\n";

csvContent+=title+"\n\n";

String[] registerKeys= registers.keySet().toArray(new String[1]);

for(int j=0; j<registers.size(); j++) {

csvContent+=registers.get(registerKeys[j]).reg.title+","+registers.get(registerKeys[j]).reg.value+","+registers.get(registerKeys[j]).reg.unit+"\n";

}

csvContent+="\n";

String[] trendKeys= dailyTrends.keySet().toArray(new String[1]);

for(int i=0; i<dailyTrends.size(); i++) {

String[] valueKeys= dailyTrends.get(trendKeys[i]).values.keySet().toArray(new String[1]);

for(int k=0; k<dailyTrends.get(trendKeys[i]).values.size(); k++) {

csvContent+=valueKeys[k]+","+dailyTrends.get(trendKeys[i]).values.get(valueKeys[k])+"\n";

}

csvContent+="\n";

}

return csvContent;

}

**Delete**

MCCU has the option to delete any file or folder within the main directory *ABBlogs*. When the user chooses to delete a new pop-up displays asking the user if they wish to confirm or cancel the deletion. If confirm is pressed, the contents are deleted. If cancel is pressed, they are brought back to the File Explorer (page-3). Files may also be deleted manually by mounting the Android device to the computer.

*Deleting any file or folder within the main directory:*

bDelete.setOnClickListener( new OnClickListener() {

public void onClick(View arg0) {

original = new File(absolutePathChild.getPath());

final String MESSAGE = "Are you sure you want to delete?" + original.getName();

AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(

context);

// set title

alertDialogBuilder.setTitle("Delete Configured Device?");

// set dialog message

alertDialogBuilder

.setMessage(MESSAGE)

.setCancelable(false)

.setPositiveButton("Confirm",new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog,int id) {

original.delete();

setFileList(absolutePath);

dialog.dismiss();

}

})

.setNegativeButton("Cancel",new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog,int id) {

dialog.cancel();

}

});

// create alert dialog

//AlertDialog alertDialog = alertDialogBuilder.create();

// show it

alertDialogBuilder.show();

optionsDialog.dismiss();

}

});

**Rename**

MCCU opens a new dialogue with a text box for the user to enter a new name. The dialogue also contains the confirm and cancel button. If the user presses cancel, they are brought back to the File Explorer page. If the user presses confirm, the entered file name is checked against a list of standard naming conventions. If the file name passes, the name is changed and the File Explorer is updated to display this change. If the name does not pass, a toast of the specific error is displayed and the user can try again.

bRename.setOnClickListener( new OnClickListener() {

public void onClick(View arg0) {

if(true) {

original = new File(absolutePathChild.getPath());

// a new file is created under that name

File destination = new File(newFileName);

// the original file is renamed to the new name

original.renameTo(destination);

optionsDialog.dismiss();

//final File dir = context.getDir(.USER\_LISTS, Context.MODE\_PRIVATE);

//final File myFile = new File(dir, filename);

AlertDialog.Builder fileDialog = new AlertDialog.Builder(context);

fileDialog.setTitle("Rename file");

// Set an EditText view to get user input

final EditText input = new EditText(context);

input.setText(original.getName());

fileDialog.setView(input);

fileDialog.setPositiveButton("Ok",

new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {

String fileName = absolutePath + "/" + input.getText().toString();

original.renameTo(new File(fileName));

setFileList(absolutePath);

}

});

fileDialog.setNegativeButton("Cancel",

new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int whichButton) {

// Canceled.

dialog.dismiss();

}

});

fileDialog.create();

fileDialog.show();

}

else {

optionsDialog.dismiss();

}

}

});

## 

## 3.5 MCCU Bluetooth Lifecycle

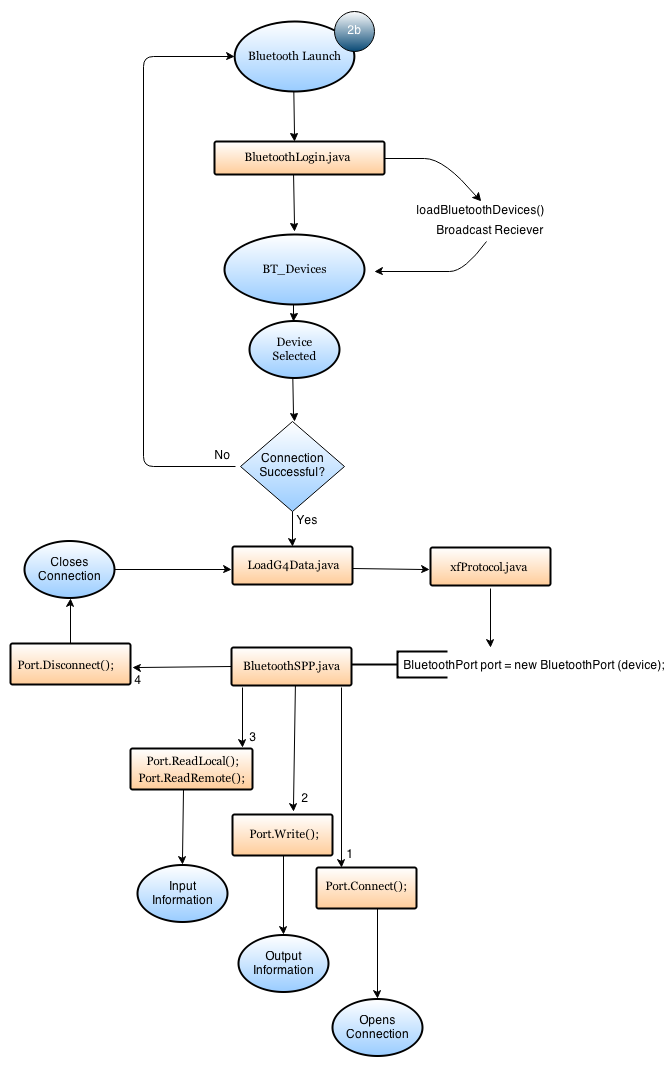


Figure 15 Bluetooth Application Lifecycle

The MCCU uses Bluetooth SPP to support connection and data request from the Totalflow G4 device (page 2a of the UI). After the user opens the Android application and chooses to connect to the G4 device using Bluetooth, the Application then checks to see if the Bluetooth setting is turned on within the Android by checking if the Bluetooth Adapter is enabled within **Home.java**:

mBluetoothAdapter.enable()

After Bluetooth has been turned on from the **BluetoothLogin.java** class, the device then scans and populates a list of Bluetooth devices within distance. This is done so by creating the Bluetooth device list:

private void loadBluetoothDevices()

The Bluetooth device list then goes through and adds the devices and their address from an array to a viewable list for the user to pick from (figure 16). When it finds a device, the MAC address and name of device will be displayed. All code coming from **BluetoothLogin.java** class:

private void loadBluetoothDevices() {

//Load the array from memory

try

{

ObjectInputStream ios = new ObjectInputStream(new FileInputStream(Environment.getExternalStorageDirectory()+"/ABBlogs/BTDevices.mccu"));

configuredDevices = (ArrayList<BT\_Devices>) ios.readObject();

//Toast.makeText(this,"LOADED LIST SUCCESSFULLY",Toast.LENGTH\_LONG);

//Log.d("DEBUG", "SUCCESS! -147");

}

catch (FileNotFoundException f)

{

configuredDevices=new ArrayList<BT\_Devices>();

// Toast.makeText(this,f.toString(),Toast.LENGTH\_LONG);

// Log.d("DEBUG", "FNF! -153");

}

catch (Exception e)

{

// Toast.makeText(this,e.toString(),Toast.LENGTH\_LONG);

// Log.d("DEBUG", e.toString()+" -158");

configuredDevices=new ArrayList<BT\_Devices>();

}

}

After the connection list has been established, the application then jumps into **RegisterLibrary.java** to begin the requests needed for data sets from different registers. Every Bluetooth socket connection requires a UUID from the “host” to the “client” used. The UUID in this application signifies. **BluetoothLogin.java** uses methods that extend from **Port.java**. This information requested will be obtained after the port connects via Bluetooth. The sockets created in ***Port.java*** use the following methods to establish the ***connect(),*** the ***write(),*** ***read(),*** ***getData(),*** and the ***disconnect().***

public abstract class Port {

public abstract void disconnect() throws IOException;

public abstract boolean connect() throws IOException;

public abstract int write(byte[] data, int numBytes) throws IOException;

public abstract int read(int numBytes) throws IOException;

public abstract byte[] getData();

}

Using the extension of **Port.java** in **BluetoothPort.java** while setting up the UUID for Bluetooth connection with G4 device:

private static final UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

private BluetoothSocket socket;

* **Setting up socket based connection to get the byte streams:**

public boolean connect() throws IOException {

if (socket== null || !socket.isConnected()) {

socket = totalflowDevice.createRfcommSocketToServiceRecord(uuid);

if (socket != null) {

socket.connect();

if (socket.isConnected()) {

outStream = socket.getOutputStream();

inStream = socket.getInputStream();

return true;

}

}

return false;

}

return true;

}

* **Setting up the Port Write to write the bytes to the output stream between the devices:**

public int write(byte[] data, int numBytes) throws IOException {

if(data.length>0 && numBytes>0) {

if(connect()) {

outStream.write(data, 0, numBytes);

return numBytes;

}

disconnect();

}

return 0;

}

* **Setting up the Port Read for Local:**

public int read(int numBytes) throws IOException {

data=new byte[numBytes];

if(this.connect()) {

int bytesRead, totalRead=0;

do {

bytesRead=0;

try {

Thread.sleep(150);

}

catch (Exception e)

{

Log.e("MCCU", e.toString());

}

if(inStream.available()>0) {

int bytesWanted = numBytes-totalRead;

bytesRead = inStream.read(data, totalRead,bytesWanted);

if(bytesRead>0) {

totalRead+=bytesRead;

if(totalRead== numBytes) {

return numBytes;

}

}

}

} while(bytesRead>0);

disconnect();

}

return 0;

}

* **Setting up the Port Disconnect to close the socket connection and the streams:**

public void disconnect() throws IOException {

if(inStream!=null) {

inStream.close();

}

if(outStream != null) {

outStream.close();

}

if(socket!=null && socket.isConnected()) {

socket.close();

}

}

* **Returning the data:**

public byte[] getData() {

return data;

}

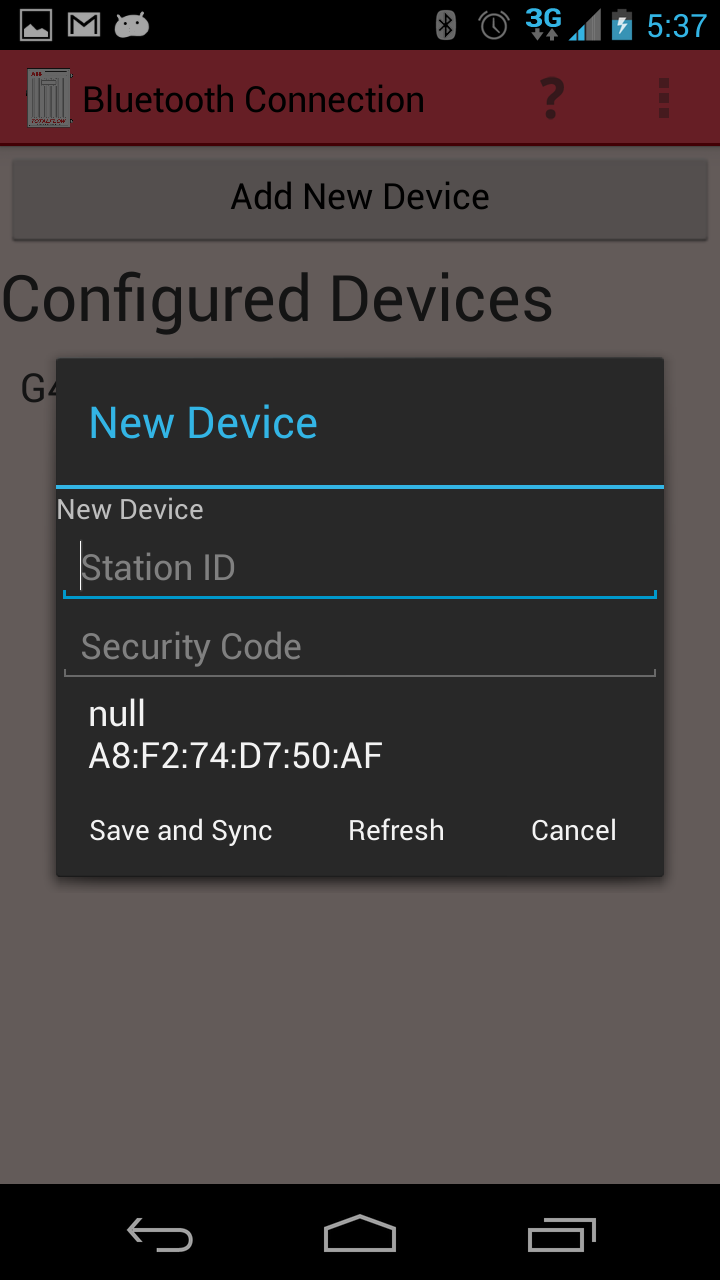


Figure 16 Screenshot of page-2a for Bluetooth log in

## 3.5.1 MCCU Bluetooth Remembering Devices (TRS2)

To enable easy access when connecting to a G4 device, the Android MCCU shall store connection information from sessions each time a new device is added. The user will be able to pick from all devices that have been previously synced by the displayed list. Once the user selects the device from the list, the necessary connection information will be loaded and the sync will begin without any input needed.

The *BT\_Devices* class is called from the functions in *BluetoothLogin()*

In the code below, there are variables created to save the input as strings from the user when connecting to a G4 device for the first time. These variables are then used for any connection made after the first time.

public class BT\_Devices implements Serializable

{

private String id;

private String password;

private transient BluetoothDevice device = null;

private String deviceAddress = "00:00:00:00:00:00";

private String deviceName = "";

public BT\_Devices(String id, String password, BluetoothDevice device)

{

this.id = id;

this.password=password;

this.device=device;

deviceAddress=device.getAddress();

deviceName=device.getName();

}

public String getID()

{

return id;

}

public String getPassword()

{

return password;

}

public BluetoothDevice getBTDevice()

{

if(device!=null)

{

return device;

}

return null;

}

public String getDeviceAddress()

{

return deviceAddress;

}

public String getDeviceName()

{

return deviceName;

}

}

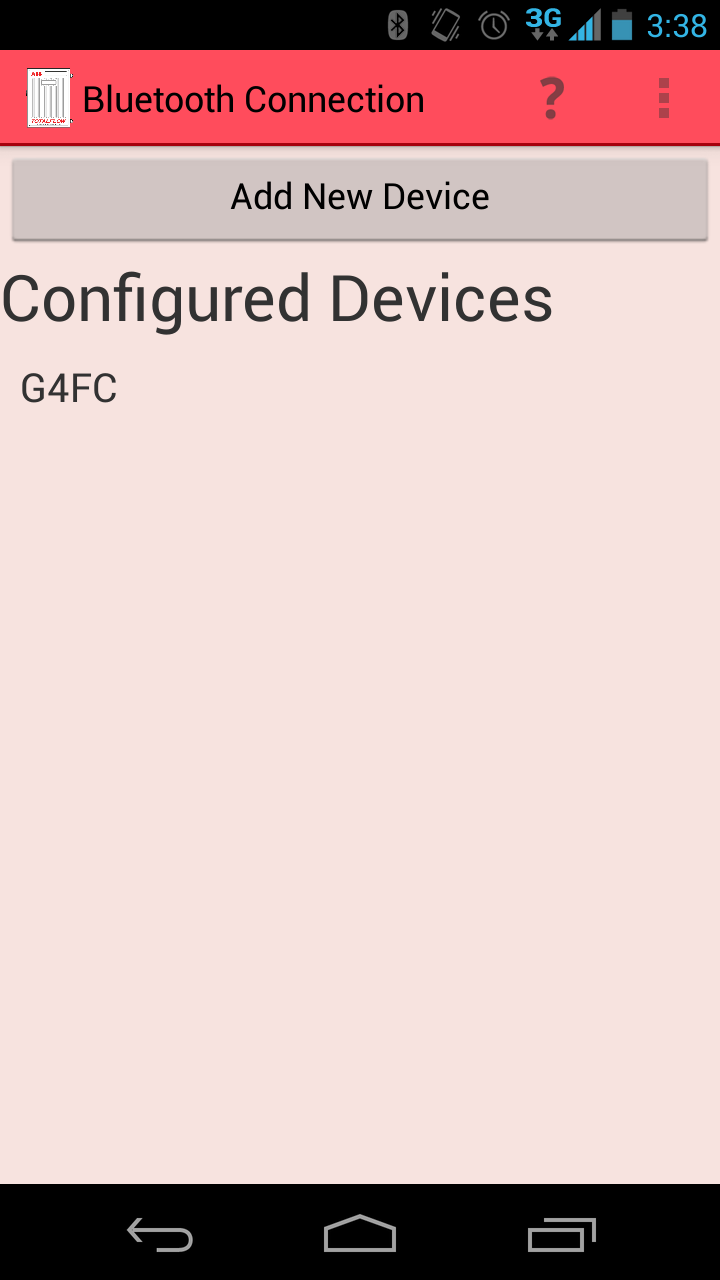


Figure X Screen shot for Bluetooth Remembered Devices

## 3.6 MCCU WiFi Lifecycle

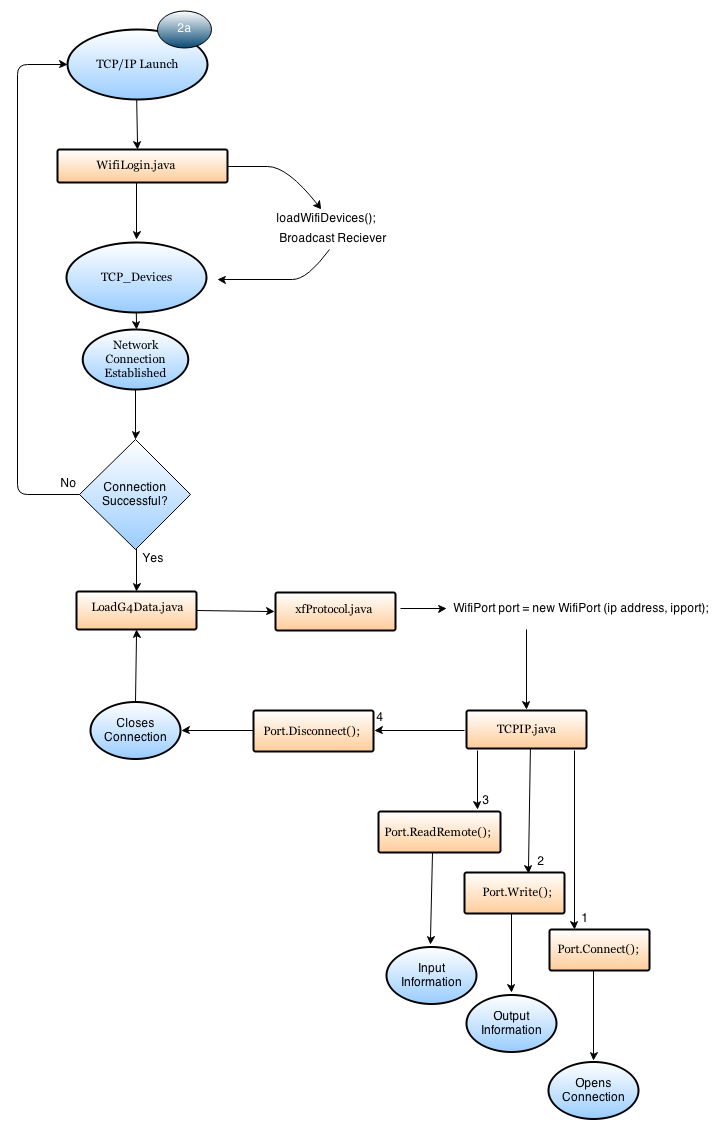


Figure 17 TCP/IP Application Lifecycle

The Android application uses TCP/IP to support connection and data request from the Totalflow G4 device (page 2b of the UI). After the user opens the Android application and chooses to connect to the G4 device using WiFi, the Application then checks to see if the WiFi setting is turned on within the Android by checking if the WiFi Adapter is enabled within **Home.java**:

mWiFiAdapter.enable()

As long as Wi-Fi or Mobile Data is turned on then user will be allowed to progress to the **WiFiLogin.java.** MCCU does not have the function to connect to a specific network within the app. The user will have to connect outside the app. Minimizing MCCU to configure Wi-Fi settings does not affect the running of MCCU. MCCU does not check if the network correctly corresponds to the network the G4 device is hosted on. If it is the wrong network, when MCCU attempts to connect to the IP address a Java Exception will be thrown and the user will be returned to the home screen.

public TCPIP(String ipAddress, String port)

this.ipAddress=ipAddress;

this.port=Integer.parseInt(port);

* **Setting up socket based connection to get the byte streams:**

**public** **byte** connect() {

**byte** ret=(**byte**)255;

**try** {

InetAddress inet = InetAddress.*getByName*(ipAddress);

*socket*= **new** Socket(inet, port);

*outStream*=*socket*.getOutputStream();

*inStream*=*socket*.getInputStream();

ret=0;

} **catch**(IOException e) {

e.printStackTrace();

disconnect();

}

**return** ret;

}

* **Setting up the Port Write to write the bytes to the output stream between the devices:**

**public** **int** Write(**byte**[] departingData, **int** nBytes) {

**int** ret;

**try** {

*outStream*.write(departingData, 0, nBytes);

*outStream*.flush();

ret=nBytes;

} **catch** (IOException e) {

disconnect();

e.printStackTrace();

ret=0;

}

**return** ret;

}

* **Setting up the Port Read for Remote to read in bytes from the input stream between the devices:**

**public** **int** ReadRemote(**int** nBytes) {

**int** index=0;

**int** totalRead=0;

**int** buf=0;

**byte**[] buffer = **new** **byte**[nBytes];

*returningData*=**new** **byte**[nBytes];

**do** {

**try** {

Thread.*sleep*(100);

buf= *inStream*.read(buffer);

totalRead+=buf;

**for**(**int** i=0; i<buf; i++) {

*returningData*[index]=buffer[i];

index++;

}

} **catch** (IOException e) {

disconnect();

totalRead=0;

**break**;

} **catch** (Exception e) {

totalRead=0;

disconnect();

**break**;

}

} **while**(totalRead<nBytes && *returningData*[19]!=0x01 && *returningData*[12]==0x00 && totalRead>=0 && *returningData*[0]==1);

**return** totalRead;

}

* **Setting up the Port Disconnect to close the socket connection and the streams:**

**public** **byte** disconnect() {

**byte** ret = (**byte**)255;

**try** {

**if**(*inStream*!=**null**) {

*inStream*.close();

} **if**(*outStream*!=**null**) {

*outStream*.close();

} **if**(*socket*!=**null**) {

*socket*.close();

}

ret=0;

} **catch**(IOException e) {

e.printStackTrace();

}

**return** ret;

}

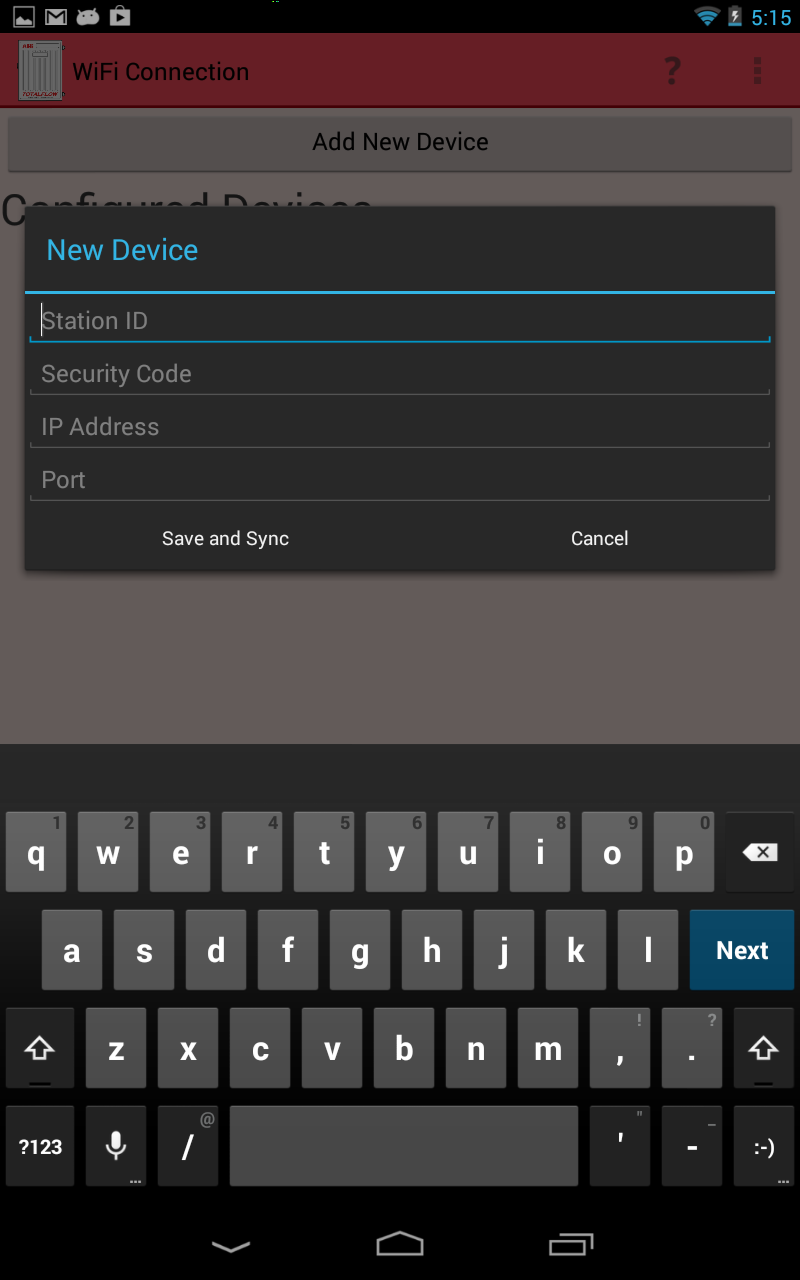


Figure 18 Screenshot of page-2b for TCP/IP connection

## 3.6.1 MCCU TCP Remember Devices (TRS2)

To enable easy access when connecting to a G4 device, the Android MCCU shall store connection information from sessions each time a new device is added. The user will be able to see all devices that have been previously synced on this list. Once the user selects the device from the list, the necessary connection information will be loaded and the sync will begin without any input needed.

The **TCP\_Devices.java** class is called from the functions in *WifiLogin()*

In the code below, there are variables created to save the input as strings from the user when connecting to a G4 device for the first time. These variables are then used for any connection made after the first time.

public class TCP\_Devices implements Serializable

{

private String password="";

private String stationID = "";

private String ipAddress="";

private String port="";

public TCP\_Devices(String password, String stationID, String ipAddress, String port)

{

this.stationID = stationID;

this.password=password;

this.ipAddress=ipAddress;

this.port=port;

}

public String getStationID()

{

return stationID;

}

public String getPassword()

{

return password;

}

public String getIpAddress()

{

return ipAddress;

}

public String getPort()

{

return port;

}

}

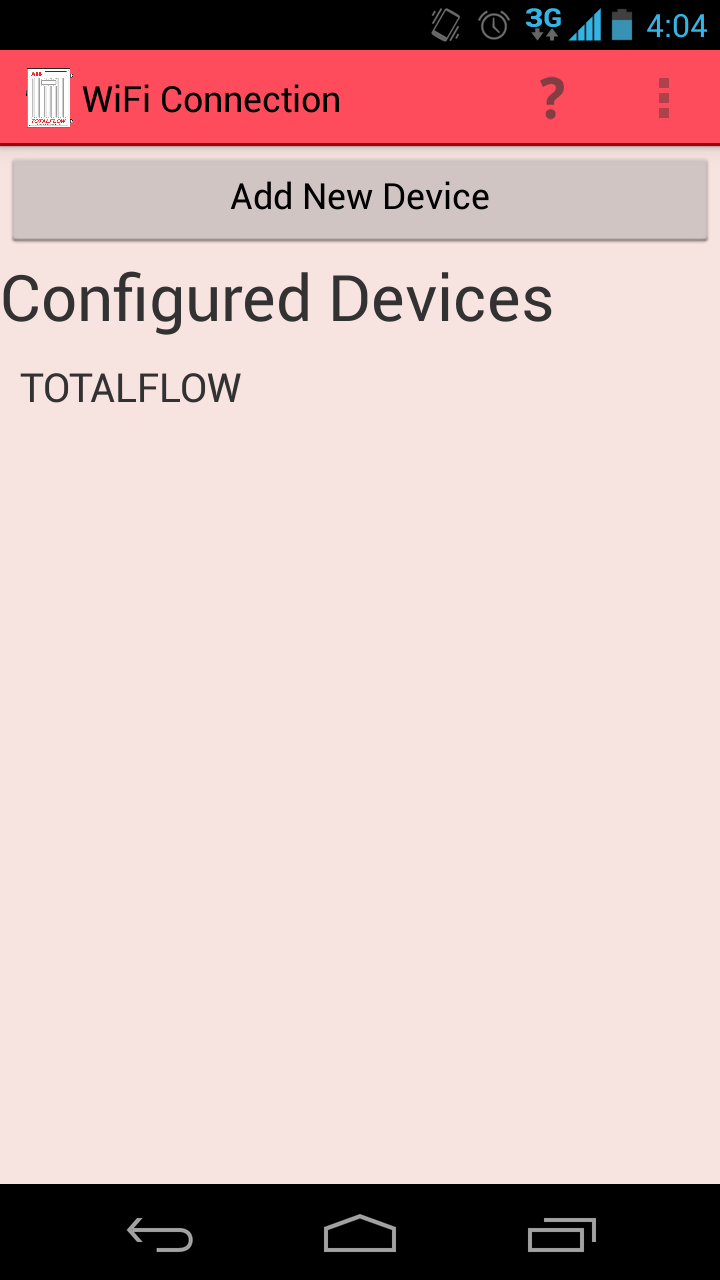


Figure X Screen shot for Remembered WiFi Devices

## 3.7 MCCU Settings (TRS6)

When displaying data on MCCU, the user may want to limit or maximize the items to be displayed based on their own user preferences. From the Home screen (*page-1*), the user will be able to access the Settings (*page-4*) to give them control of the data that is displayed in Data View (*page-6*). The user shall first select their application to edit the Data View for by sliding out the Application drawer from the left side of the screen. The selection of items to be edited is created in an expandable list, allowing the user to select what type of measurement they want to edit. The expandable adaptor is built of the base list for the expandable adaptor is an inner, meaning it cannot be re-used. This is another page that utilizes **AppView.java** class to create the list of expandable items. This allows the list to populate based upon the tube type that the user selects from the Application Selection.

At this time there are no other options available in the settings page, however, all future settings can be built here and implemented into the **Settings.java** class.

Code below shows how the drawer is activated for the Settings page:

mDrawerToggle = new ActionBarDrawerToggle(

this, /\* host Activity \*/

mDrawerLayout, /\* DrawerLayout object \*/

R.drawable.ic\_drawer, /\* nav drawer image to replace 'Up' caret \*/

R.string.drawer\_open, /\* "open drawer" description for accessibility \*/

R.string.drawer\_close /\* "close drawer" description for accessibility \*/

) {

public void onDrawerClosed(View view) {

getActionBar().setTitle(mTitle);

invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()

}

public void onDrawerOpened(View drawerView) {

getActionBar().setTitle(mDrawerTitle);

invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()

}

};

mDrawerLayout.setDrawerListener(mDrawerToggle);

if (savedInstanceState == null) {

selectItem(0);

}

Code below shows how the options can be selected to the user preference:

public static class PlanetFragment extends Fragment {

public static final String ARG\_PLANET\_NUMBER = "planet\_number";

private ListView mainListView;

private mItems[] itemss;

private ArrayAdapter<mItems> listAdapter;

ArrayList<String> checked = new ArrayList<String>();

Context context;

public PlanetFragment() {

// Empty constructor required for fragment subclasses

}

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container,

Bundle savedInstanceState) {

context = getActivity();

View rootView = inflater.inflate(R.layout.setting\_list, container, false);

int i = getArguments().getInt(ARG\_PLANET\_NUMBER);

//Inset stuff Here

mainListView = (ListView) rootView.findViewById(R.id.mainListView);

//inputSearch = (EditText) findViewById(R.id.inputSearch);

// When item is tapped, toggle checked properties of CheckBox and

// Planet.

mainListView

.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View item,

int position, long id) {

mItems planet = listAdapter.getItem(position);

planet.toggleChecked();

SelectViewHolder viewHolder = (SelectViewHolder) item.getTag();

viewHolder.getCheckBox().setChecked(planet.isChecked());

//Toast.makeText(context, "I am Checked, "+viewHolder.toString(),Toast.LENGTH\_LONG).show();

}

});

// Create and populate planets.

// itemss = (mItems[]) getLastNonConfigurationInstance();

ArrayList<mItems> planetList = new ArrayList<mItems>();

planetList.add(new mItems("Volume Flow Rate"));

planetList.add(new mItems("Today's Volume"));

planetList.add(new mItems("Yesterday's Volume"));

planetList.add(new mItems("Accumulated Volume"));

planetList.add(new mItems("Last Calc Period Volume"));

planetList.add(new mItems("Diff. Pressure"));

planetList.add(new mItems("Static Pressure"));

planetList.add(new mItems("Energy Rate"));

// Set our custom array adapter as the ListView's adapter.

listAdapter = new SelectArralAdapter(context, planetList);

mainListView.setAdapter(listAdapter);

//getActivity().setTitle(planet);

return rootView;

}

}

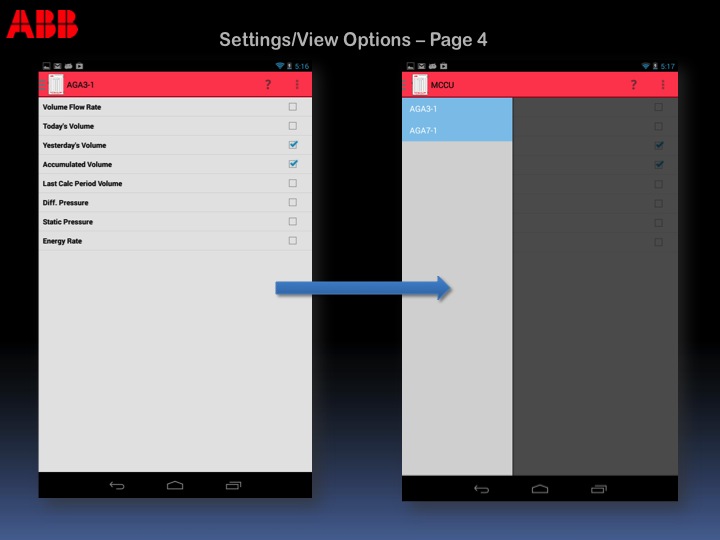


Figure x screenshot for Settings page with Application Drawer

## 3.8 Design Aspects for Application Performance

The MCCU application will be used to have limited resources in terms of memory and CPU. In order to achieve the time for individual page loading to around one second and comparable to the time taken by similar screens on MCCU, following parameters need to be considered. The other aspect that needs to be considered is the hardware capability of the Android device used to run the application.

The Action Bar items such as Help and the Action Bar Loader are used to make the application performance more efficient. The User will not have to navigate backwards on some pages just to pull up Help options or to get the Settings page for Data View options.

## 3.8.1 Page Size and Corresponding Graphics

Number of graphics and size of all graphics on a page have a direct impact on the response time for that application page. To improve the performance of any application page, the quality and number of graphics needs to be minimized. In order to reduce the size of the graphics used on any application page there are some techniques like compressing the content, reducing the image sizes to the desired resolution. By placing multiple .PNG files with varying pixel counts, according to size in different folders, the application gains efficiency.

## 3.8.2 Number of Threads

The Android application has one thread dedicated to the user interface. The Android OS forces all network and Bluetooth communications onto a separate thread. A new thread is created when the Android device begins receiving register data from the G4. After register data is obtained, the thread is finished. If the transmission is canceled then an error is forced into the protocol causing it to finish the thread early in order to maximize performance. A thread is created on pages 2a and 2b to sync data from the G4 device. A progress dialog spinner is displayed on the UI thread while the other thread finishes it course to indicate the user that a sync is being made (Section 3.7.3).



Figure 20 Spinner displayed to represent thread for obtaining data

## 3.8.3 Indicative/Responsive UI

To ensure that user is always aware of the UI processing and events feedback, progress indicators are used so that user knows that processing is in progress and screen is not in a frozen/blocked state.

After the user inputs their information for connecting to a device via Bluetooth or IP, the application will display the dialogue spinner to let the user know that their action is being processed.

*Spinner being created in BluetoothMain.java class:*

private class ProgressTask extends AsyncTask <Void,Void,Void> {

@Override

protected void onPreExecute(){

mProgressDialog.setVisibility(View.VISIBLE);

}

****

Figure 21 Screenshot example for “spinner” used when loading data

Android OS has built in identifiers to indicate Wifi and Bluetooth status. Refer to section 3.3 for indicator information.

## 3.8.4 Recovery from crash/hang

If the Android application crashes due to some problem/bug and causes Android MCCU to exit; the user must re-open the Android application and start again at the Home page (page-1). If the application crashes during Sync (page-2) or during the storing of a file, the data is not stored on the Android device. If the application crashes during Options View (page-4a), the user preferences are not saved. If the Android device does not get a response from the G4 device, it will throw a timeout error. Any other crashes during navigation or operation shall bring the user back to the Home page (page-1).

## 3.8.5 Memory Usage

The Android application uses internal and external memory to maximize application performance. When navigating from page to page, the previous page is stored as a variable. When the user presses the “back” button to navigate to the previous page, that variable will be accessed. This functionality is shown as follows:

The use of stacks is implemented into memory for the MCCU application by the Android SDK libraries. All of Navigation throughout the MCCU application is taken care of by the Devices internal memory and Android’s OS for handling pages.

For File Manager, refer to section 3.4.2.

## 3.9 Register Request Protocol (TRS4)

Only the Android device accesses the predefined registers when the user connects to the G4 device (TRS4). All of the register types are read and write based and are defined in the format *app.array.index*. The data can be stored into the Android device’s external memory to be shared or to be displayed (TRS4) in lists and charts from the device. The below code is only an example of the AGA3 application, all other applications follow same format of data request. All application request made can be found in **RegisterLibrary.java**.

Requests are sent out in the format (*Application*, *Array*, *Index*, *# of registers*, *data size (in bytes)*).

public ArrayList<Application> getCorrectApps\_AddRegisters(xfProtocol xf) throws IOException {

boolean appFound = false;

ArrayList<Application> apps = new ArrayList<Application>();

The first part loops through looking for the enumerated identifiers. If an enumeration is found the slot number is added to its corresponding list:

byte dataEnum;

for (int i = 0; i < xf.getDataIndex(); i++) {

dataEnum = xf.getDataElement(i);

switch (dataEnum) {

case 4: //aga3

Application aga3 = new Application(dataEnum, i, "AGA3");

aga3.registers.put(i+".5.0", new xfRequest(new Register(i,5,0, "Tube Name", false, "", "", 4, false), 1, 65));

aga3.registers.put(i+".0.42", new xfRequest(new Register(i,0,42, "VolumeFlowRateUnit", true, "", "\*VolumeFlowRate", 3, false), 1, Byte.SIZE/8));

aga3.registers.put(i+".7.19", new xfRequest(new Register(i,7,19, "Current Flow Rate", false, "", i+".0.42\_FlowRate", 0, false), 1, Float.SIZE/8));

aga3.registers.put(i+".7.22", new xfRequest(new Register(i,7,22, "Today's Volume", false, "", i+".0.42\_Volume", 0, false), 1, Float.SIZE/8));

aga3.registers.put(i+".7.23", new xfRequest(new Register(i,7,23, "Yesterday's Volume", false, "", i+".0.42\_Volume", 0, false), 1, Float.SIZE/8));

aga3.registers.put(i+".3.73", new xfRequest(new Register(i,3,73, "Current Energy Rate", false, "", "EnergyRate", 0, false), 1, Float.SIZE/8));

aga3.registers.put(i+".3.71", new xfRequest(new Register(i,3,71, "Today's Energy", false, "", "Energy", 0, false), 1, Float.SIZE/8));

aga3.registers.put(i+".3.72", new xfRequest(new Register(i,3,72, "Today's Energy", false, "", "Energy", 0, false), 1, Float.SIZE/8));

aga3.registers.put(i+".3.3", new xfRequest(new Register(i,7,22, "Temperature", false, "", "Temperature", 0, false), 1, Float .SIZE/8));

aga3.registers.put(i+".7.0", new xfRequest(new Register(i,7,22, "Differential Pressure", false, "", "DifferentialPressure", 0, false), 1, Float.SIZE/8));

aga3.registers.put(i+".3.0", new xfRequest(new Register(i,7,22, "Static Pressure", false, "", "StaticPressure", 0, false), 1, Float.SIZE/8));

apps.add(aga3);

appFound=true;

break;

## 3.10 Register Grouping

Registers do not have to be grouped in any orderly fashion when sending a request. After the data has returned for the group of register requests, MCCU categorizes the data for a user-friendly display.

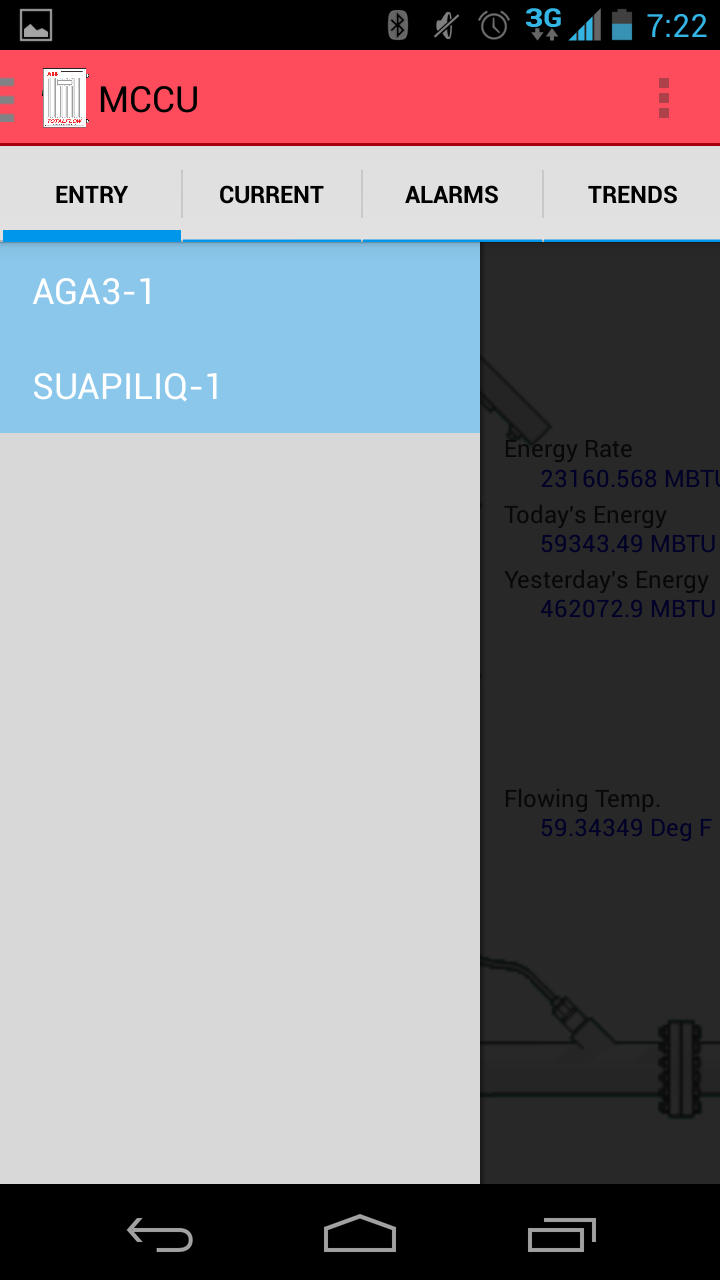


Figure 22 Application Selection Page

## 3.10.1 App Data Selection

When the user views a set of data from the G4 device, they will obtain a list of applications from the left slide out drawer (Page 4). By clicking on a member of that list the following tabs will be filled with its corresponding information. The user must refresh from MCCU application to further update their data taken from the G4 device.

The list of tubes derives from the instances of tube applications on the device. It is one tube per application. The list of tanks derives from not only the instances of level master applications but the number of tanks per level master application. The number of tanks within a LevelMaster application has to be figured first through a separate register request.

Tube and Tank CODE

Tanks in Drawers Image

Figure 23 Screen shots for tank selection

## 3.10.2 Entry Mode

The General Data covers a range of information that is displayed simply as:

‘*category description:’ ‘retrieved data’ ‘unit’ (if applicable).* See figure below.

For **Tubes**, general data is specifically set to display flow measurement. When reading flow measurement from the G4 device, the user will be able to see current data as well as the entire day, yesterday, and accumulated for both gas and liquids. Flow measurement categories include: volume, flow rate, energy, energy rate, static pressure, differential pressure, temperature, pulse, and mass. When reading flow measurement from the G4 device, the user will be able to see all data with corresponding units of measurement. Since the units can be changed for certain registers, additional register request must be made to get the correct unit.

Units register requests made from **Application.java** class using three methods: *addRegisterUnit(), identifyRegisterUnits(), and setRegisterUnits().* See code below for Unit methods:

public void addRegisterUnit(int appEnum, Register unitRegister) {

Unit tempUnit = new Unit(unitRegister, appEnum);

tempUnit.format();

unitRegisters.put(unitRegister.unitGroup.substring(1), tempUnit);

}

public void identifyRegisterUnits() {

// gathers all of the keys

String[] registerKeys= registers.keySet().toArray(new String[1]);

// for each key

for(int i=0; i<registerKeys.length; i++) {

// get the Register and its unitGroup

Register currentRegister=registers.get(registerKeys[i]).reg;

// based on the unitGroup update the unit for the Register

if(currentRegister.unitGroup.startsWith("\*")) {

addRegisterUnit(APPLICATION, currentRegister);

registers.remove(registerKeys[i]);

}

}

}

public void setRegisterUnits() {

// gathers all of the keys

String[] registerKeys= registers.keySet().toArray(new String[1]);

// for each key

for(int i=0; i<registers.size(); i++) {

// get the Register and its unitGroup

String unitGroup = registers.get(registerKeys[i]).reg.unitGroup;

if(unitGroup.contains("\_")) {

String subUnitGroup=unitGroup.substring(unitGroup.indexOf("\_")+1);

unitGroup=unitGroup.substring(0, unitGroup.indexOf('\_'));

try {

registers.get(registerKeys[i]).reg.unit= unitRegisters.get(unitGroup).values.get(subUnitGroup);

} catch(NullPointerException e) {

registers.get(registerKeys[i]).reg.unit= "";

}

continue;

}

try {

registers.get(registerKeys[i]).reg.unit= unitRegisters.get(unitGroup).values.get(unitGroup);

} catch(NullPointerException e) {

registers.get(registerKeys[i]).reg.unit= "";

}

}

}

Figure 24 Screenshot example for Flow Measurement Data collected

For **Tanks**, two tabs are used to display the general data for the device. The setup/status tab shows information regarding tank identification and polls. The levels tab shows information regarding tank measurements. Tank measurement categories include: level volume, level inches, temperature, and Ullage. The units for LevelMaster are static.

Figure 25 Screenshot example of Setup/Status Tab Data

## 3.10.3 Current Data

The Current Data tab allows the user to provide input to the G4 from their Android device. The user will be able to long click one of the data values on the screen and a pop-up will prompt the user with an input field allowing them to edit the value for that data point. The **Data\_View.java** class contains all of the methods for creating the page seen when selecting the Current Data tab. The data is displayed as a grid table with three rows and two columns. The code below shows the three methods used to create the grid as well as setting up the data and the view within:

public void createAGA3Table(View rootView)

{

GridView gridView;

ArrayList<Item> gridArray = new ArrayList<Item>();

CustomGridViewAdapter customGridAdapter;

gridView = (GridView)rootView.findViewById(R.id.gridView1);

//gridView = (GridView)rootView.findViewById(R.layout.dataview\_testing);

String[] registerKeys = app.registers.keySet().toArray(new String[1]);

int i =0;

for(int z = 0;z<app.registers.size();z++)

{

gridArray.add(new Item(app.registers.get(registerKeys[z]).reg.title,app.registers.get(registerKeys[z]).reg.value,app.registers.get(registerKeys[z]).reg.unit));

}

//gridView = (GridView) findViewById(R.id.gridView1);

customGridAdapter = new CustomGridViewAdapter(AppContext, R.layout.gridtest\_item, gridArray);

gridView.setAdapter(customGridAdapter);

}

public class CustomGridViewAdapter extends ArrayAdapter<Item> {

Context context;

int layoutResourceId;

ArrayList<Item> data = new ArrayList<Item>();

public CustomGridViewAdapter(Context context, int layoutResourceId,

ArrayList<Item> data) {

super(context, layoutResourceId, data);

this.layoutResourceId = layoutResourceId;

this.context = context;

this.data = data;

}

@Override

public View getView(int position, View convertView, ViewGroup parent) {

View row = convertView;

RecordHolder holder = null;

if (row == null) {

LayoutInflater inflater = ((Activity) context).getLayoutInflater();

row = inflater.inflate(layoutResourceId, parent, false);

holder = new RecordHolder();

holder.title = (TextView) row.findViewById(R.id.title);

//holder.imageItem = (ImageView) row.findViewById(R.id.item\_image);

holder.data = (TextView) row.findViewById(R.id.data);

holder.unit = (TextView) row.findViewById(R.id.unit);

row.setTag(holder);

} else {

holder = (RecordHolder) row.getTag();

}

Item item = data.get(position);

holder.title.setText(item.getTitle());

holder.data.setText(item.getData());

holder.unit.setText(item.getUnit());

//holder.imageItem.setImageBitmap(item.getImage());

return row;

}

}

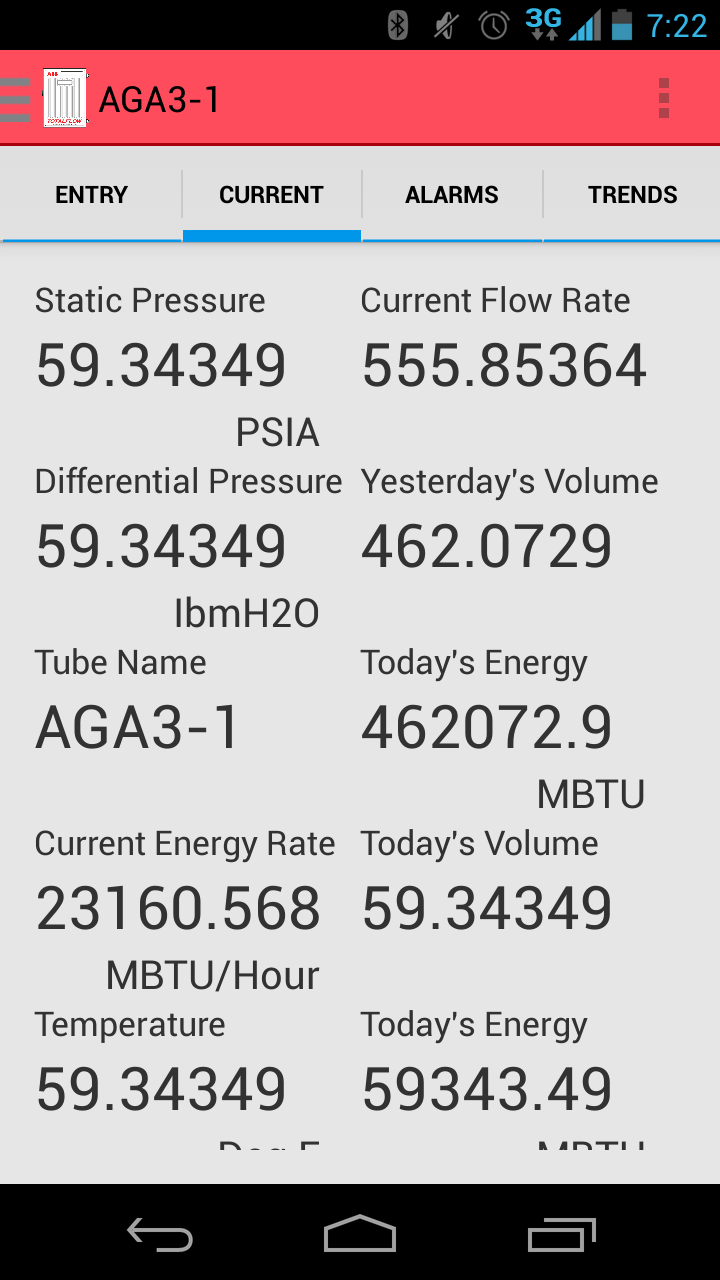
******

Figure 26 Screenshot example of Current Data with Units

## 3.10.4 Active Alarms

When reading active alarms from the G4 device, the user will be able to see any alarms. This will not allow the user to reset the alarm from the Android device. After the alarm has been reset on the G4 device, and the user re-syncs the Android application to the G4 device, the alarms will no longer show up.

The MCCU application is enhanced to provide the list of current Active Alarms based on the specific application. A register request corresponding to the Tubes current alarms is packaged with the other registers for that data transaction. The response is decoded by taking the 32-bit integer returned and converting it into binary. Each bit correlates to a specific alarm within a set of 20 total alarms. If the bit is equal to 0, the alarm is off. If the bit is equal to 1, the alarm is on. The two character code that corresponds to that bit is shown if it is on.

A register request corresponding to the tank current errors and warnings are packaged with the other registers for that data transaction. The response corresponds to a single error in an enumeration of errors/warnings.

**Image and Code for Alarms tab**

Figure 27 Screenshot of Alarm Tab Data

## Alarm Log

## MCCU does not provide for an updated list of Alarm logs, only current active alarms. Any alarm viewed from the application is an active alarm request. All alarms that are viewed and unacknowledged can be saved as a file into the external thread and then viewed by the user later. Essentially the user is creating their log of alarms.

**3.10.5 Trends**

Trend data is gathered with a single register request. The G4 device has designated arrays that contain historical data for certain categories.

**NEED CODE AND IMAGE FOR TRENDS**

Figure 28 Screenshot of Trends Tab Data

Next to the category name that contains the past 14 days is a button that will launch the user to an interactive bar graph. The graphs are used to visualize the various parameters and monitor them over a period of time. There will be X and Y axis which will represent the parameters that are being compared against each other – measured data being the Y axis and duration of time being the X axis. A legend is also shown to describe multiple parameters in case multiple parameters are being used in the graph.

For implementing the graph in Trends screens, ‘ChartEngineDemo’ library is used.

The following is called from the TrendsBarChart.java class:

//Adding color to be used for the Bars

**int**[] colors = **new** **int**[] {0xffff6c00};

XYMultipleSeriesRenderer renderer = buildBarRenderer(colors);

//Adding Renderer Style, chart title, labels, X-axis boundaries, Y-axis boundaries, and label text color

setChartSettings(renderer, "Historical Data for the past 14 days", "Day", " "+floatUnit, 0.5,

7.5, 0, max, 0xff989898, 0xff989898);

renderer.getSeriesRendererAt(0).setDisplayChartValues(**true**); //Adds values to the Bars

renderer.~~setChartValuesTextSize~~(22); //Changes text size of the Bar values

renderer.setLabelsTextSize(18); //Changes text size of the Axis labels

renderer.setXLabels(7); //Sets spacing for X labels

renderer.setYLabels(5); //Sets spacing for Y labels

renderer.setXLabelsAlign(Align.*LEFT*);

renderer.setYLabelsAlign(Align.*LEFT*);

renderer.setYLabelsColor(0,0xff989898); //Sets label colors to Grey

renderer.setXLabelsPadding((**float**)8); //Shifts the X-axis labels down from chart

renderer.setYLabelsPadding((**float**)30); //Shifts the Y-labels left

renderer.setPanEnabled(**true**, **false**); //allows panning along the X-axis

renderer.setPanLimits(**new** **double**[] {0.5, 14.7, 0, max}); //Limits X-axis panning to 14.7 days

renderer.setZoomEnabled(**false**);

renderer.setZoomEnabled(**false**, **false**);

renderer.setBarWidth(45); //Changes the width of each bar

renderer.setLegendTextSize(28); //Changes legend text size

//Utility method for creating chart views or intents

**return** ChartFactory.*getBarChartIntent*(context, buildBarDataset(titles, values), renderer,

Type.*STACKED*);

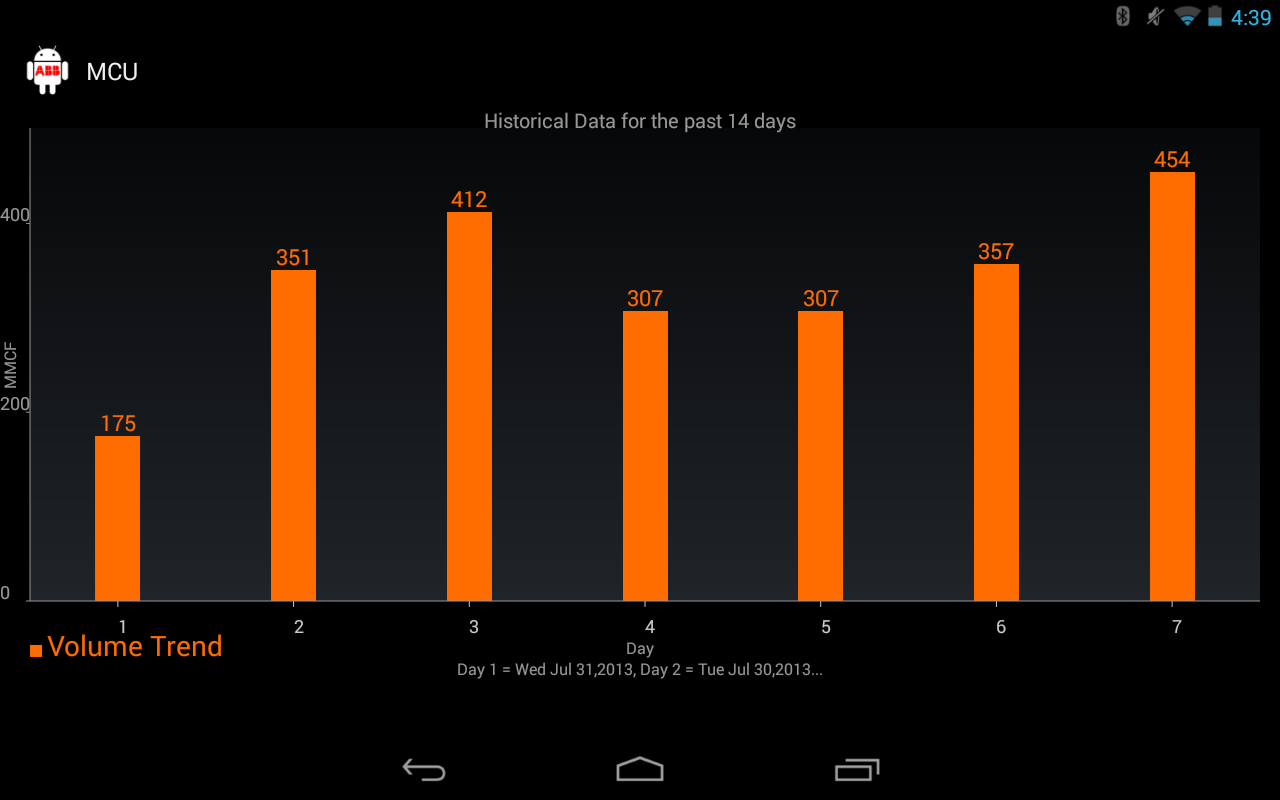
****

Figure 29 Screenshot for an example of historical data

## 3.11 Error Codes

When using MCCU application there are cases where the application may not complete a request made by the user. MCCU is designed to notify the user when there is an error during connection. When an error is caught, MCCU runs through the following cases to match the error code with a description. The user is brought back to the connection selection (page 1), and the error is displayed over the page.

private String getErrorMessage(int error) {

switch (error) {

case 200:

return error + ": Unable to locate applications for tubes.";

case 201:

return error + ": Unable to save and retrieve data.";

case 203:

return error + ": Data request cancelled.";

case 102:

return error + ": The device received a bad xfHeader CRC.";

case 104:

return error + ": The device received a bad xfRecord CRC.";

case 106:

return error + ": The device received an invalid operation request.";

case 126:

return error + ": Illegal register read or write.";

case (byte) 0x81:

return error + ": Undefined error.";

case (byte) 0x88:

return error + ": xfFrame already in use.";

case (byte) 0x96:

return error + ": Invalid application slot.";

case (byte) 251:

return error + ": xfLENERR";

case (byte) 252:

return error + ": xfCRCERR";

case (byte) 253:

return error + ": xfNAKCODE";

case (byte) 254:

return error + ": xfINVALID";

case (byte) 255:

return error + ": xfTIMEOUT";

case (byte) 207:

return error + ": No files";

case (byte) 206:

return error + ": No Bluetooth ports found on device.";

case (byte) 205:

return error + ": Connection to device failed.";

case (byte) 204:

return error + ": Unable to change communication protocol.";

default:

return error + ": Unknown error number was returned.";

}

}

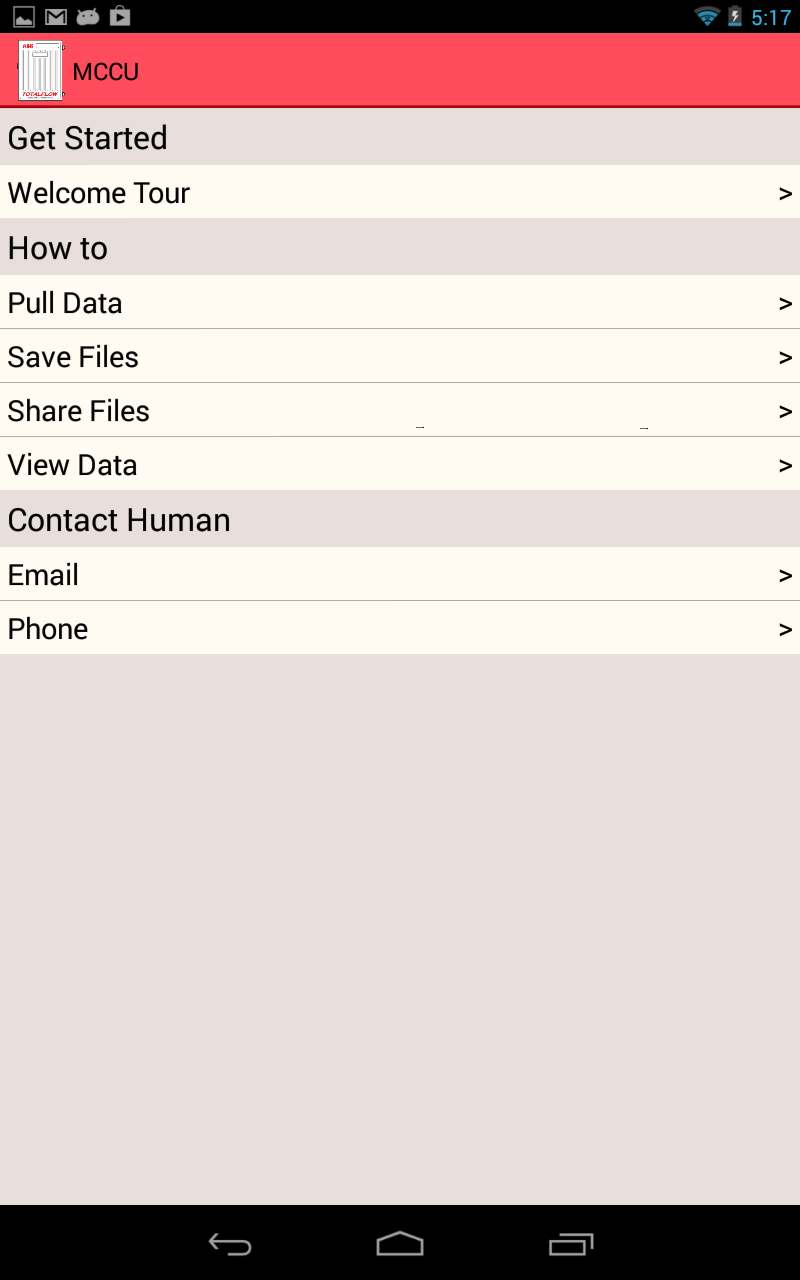
**3.11.1** **No Data from Register**

In case the device register has no value at a particular moment and the request to get the data from device register returns no data, user is prompted with pop-up dialog box displaying ‘xfCRCERR’. This behavior follows the Error Scenario Handling and all the error codes mentioned in section 3.10.

*Possible change message*

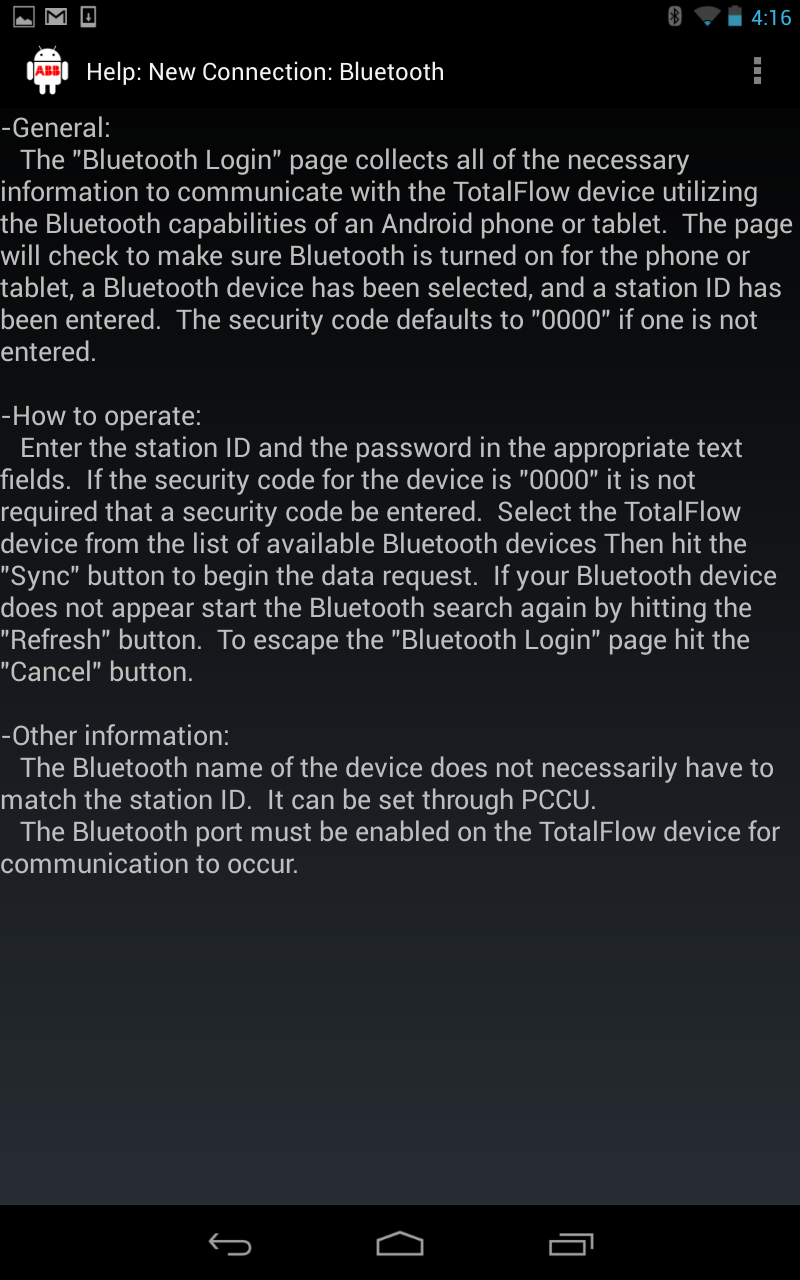
## 3.12 Help (TRS4, TRS5)

Help options are available within the Action Bar (top of screen). The help page will allow the user to navigate information on topics regarding connection for Bluetooth/IP, saving files, data view, File Explorer, Settings, and navigation.



Update Image

Figure 31 Screenshot from Help menu:



Update Image

Figure 32 Screenshot example of selecting a Help option

## 3.13 Formatting

MCCU follows the NGHLA guidelines for specific colors, fonts, and layouts used on different application pages. The built in Android theme, Holo, has been modified slightly to meet the NGHLA coloring format. When applying styles that only modify a View’s text properties the style is applied via the Text Appearance property. Styles that modify non-text properties are applied via the *style* property. By default, the Android Holo theme uses TextAppearanceMedium which is scalable based on the type of Android device and the pixel count for the screen size. The user can edit these options in the settings of the Android Device.

Holo Theme Content:

* Screen Background Dark: Black (#000000)
* Text Size: 18 scaled pixels
* Text Font: Holo

## 3.13.1 Fonts

In general, Holo Font, default theme is used as the ‘font’ across the screens to display various text. On all screens for non-value labels, font is non-bold. For Trends tab, MonoSpace font is used to make all of the characters the same width. Using MonoSpace makes the Trends dates organized so each day is lined up and the user can scan down the list of days with ease.

Custom Trends Content:

* Text Scale: 1.0
* Text Size: 20 scaled pixels
* Text Font: MONOSPACE

## 3.13.2 Error Scenario Handling

In case there is an error while saving data or running through the protocol then an error message is displayed to the user. The error message takes into account all of the format and font settings for the Android Holo Theme. This error message displays brief error description along with possible recovery step (if required). This error message helps the user to be aware of any possible error and then recover from it (section 3.10).

## 3.13.3 Res Folder

Styling information for the user interface part is maintained as separate files. There are two ways to set a style:

* To an individual View, by adding the *style* attribute to a View element in the XML for the layout.
* To an entire Activity or application, by adding the *android:theme* attribute to the <activity> or <application> element in the Android manifest.

All such information is maintained as XML files, which are applied as required at multiple places. This also ensures that styling information is not duplicated. A lot of the features on MCCU utilize the *android:theme* attribute.

## 3.13.4 Action Bar Overflow

The Action Bar Overflow appears at the top of an activity’s window when the activity uses one of the system’s descendant themes that are set by default. The Action Bar Overflow is populated with various tabs to allow the user for navigating throughout the application.

Holo Theme Content:

* Screen Background Dark: Black (#000000)
* Text Size: 18 scaled pixels
* Text Font: Holo

## 3.13.5 Tab Navigation

User can choose to browse across various screen sections using the different Tabs. Every tab contains the name that describes the content on screen. The name changes color when user clicks on that particular tab name.

Following are the key states of the tab names section on screen:-

1. Selected (This tab is the currently selected tab and displays the content that belong to that screen).
2. Un-Selected (This tab is not the active tab)

This navigation section conforms to following specification:-

1. Un-Selected

* Font Style: Holo
* Font Size: 18 scaled pixels
* Width : Theme Default
* Height: Theme Default
* Font Color: Light Grey (#c8c8c8)
* Background Color: Very Dark Grey (#666666)

1. Selected

* Font Style: Holo
* Font Size: 18 scaled pixels
* Width : Theme Default
* Height: Theme Default
* Font Color: Black (#00000)
* Background Color: Very Light Grey (#d4d4d4)

## 3.13.6 Expandable List View

The Data View page contains the Trends Tab with data that is organized in Expandable lists. The Expandable list is a view control that allows the user to provide multiple panes and display them when user clicks the list. It is like having several [panels](http://www.asp.net/ajaxlibrary/AjaxControlToolkitSampleSite/CollapsiblePanel/CollapsiblePanel.aspx) where every Expandable lists can be expanded on user click. Every Expandable list represents a logical grouping of the data. Expandable lists make it easy to display large number of data in a limited screen space. Expandable lists will conform to following specification:-

Expandable lists Section Content

* Font Style: Holo
* Font Size: 18 Scaled Pixel
* Font Color: Very Light Grey (#d4d4d4)
* Background Color: Very Dark Grey (#222222)

## 3.13.7 Graphs

The graphs used for representing trends are controlled by the ‘renderer’ class. The labels, panning, zoom, bar width, legend, text, and colors are all set here. The coloring follows the same guidelines mentioned for charts in the ABB\_NGHLA\_Layout\_Specs\_220812.pdf document.

Expandable lists Section Content

* Font Style: Holo
* Background Color: Very Dark Grey (#666666)
* Bar Label Text Size: 22
* X Label Text Size: 7
* Y Label Text Size: 5
* Y Label Color: Light Grey (#989898)
* X Label Color: Light Grey (#989898)
* Bar Width: 45
* Legend Text Size: 28