Gate 3 - Design Document Approval and Gate Participants

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

Date(s): *Summer 2013*

Project Manager: *Blaine Tiernan*

Prepared by: *Lee Easton*

Document status: \_**X**\_ Draft \_\_ Proposed \_\_ Validated \_\_ Approved

**Change Log**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Changes Description** | **Author** | **Date** |
| 1 | First draft | L Easton | 2013,June 10 |

Contents

[NOTE 3](#_Toc361898772)

[1. Introduction 3](#_Toc361898773)

[1.1 Intended Audience 3](#_Toc361898774)

[1.2 References 3](#_Toc361898775)

[1.3 Acronyms and Abbreviations 3](#_Toc361898776)

[1.4 Assumptions 4](#_Toc361898777)

[2. System Design 4](#_Toc361898778)

[2.1 System Overview 5](#_Toc361898779)

[2.1.1 Development of Totalflow Android Application (*MRS1*) 5](#_Toc361898780)

[2.1.2 Development of Remote Access App pages 5](#_Toc361898781)

[2.1.3 Development of Adaptation layer 5](#_Toc361898782)

[2.1.4 Design for application page development for test automation 5](#_Toc361898783)

[2.1.5 MCCU.apk (Flash) Changes 5](#_Toc361898784)

[2.2 Deployment Diagram 6](#_Toc361898785)

[2.3 Solution Architecture (TRS1, TRS2) 6](#_Toc361898786)

[2.4 UI Information Model (TRS4, TRS5) 8](#_Toc361898787)

[2.5 UI Screens Summary 12](#_Toc361898788)

[2.5.1 Phase 1 Screens 12](#_Toc361898789)

[3. Totalflow Android Application Design 13](#_Toc361898790)

[3.1 Application Design Overview 13](#_Toc361898791)

[3.2 Totalflow Android Application Modules 14](#_Toc361898792)

[3.2.1 Session Manager 14](#_Toc361898793)

[3.2.2 File Manager 15](#_Toc361898794)

[3.2.3 Totalflow Android Application Lifecycle 16](#_Toc361898795)

[3.4 Totalflow Android IP Lifecycle (*TRS2*) 22](#_Toc361898797)

[3.5 Design Aspects for Application Performance 26](#_Toc361898798)

[3.5.1 Page Size and Number of Objects 26](#_Toc361898799)

[3.5.2 Number of Threads 26](#_Toc361898800)

[3.5.3 Memory Usage 26](#_Toc361898801)

[3.5.4 Recovery from crash/hang 28](#_Toc361898802)

[3.5.5 Use of MVC Architecture 28](#_Toc361898803)

[3.5.6 Indicative/Responsive UI 28](#_Toc361898804)

[3.6 Registers (TRS4) 28](#_Toc361898805)

[3.6.1 Active Alarms 28](#_Toc361898806)

[3.6.2 Alarm Log 29](#_Toc361898807)

[3.6.3 General Descriptions 29](#_Toc361898809)

[3.6.4 Flow Measurement 30](#_Toc361898810)

[3.6.5 Units 31](#_Toc361898811)

[3.6.6 Level Master 32](#_Toc361898812)

[3.6.7 Warning and Error Codes 34](#_Toc361898813)

[3.7 Templates 35](#_Toc361898814)

[3.7.1 Template Guideline 36](#_Toc361898815)

[3.7.2 Res Folder 36](#_Toc361898816)

[3.7.3 Action Bar 36](#_Toc361898817)

[3.7.4 Tab Navigation 37](#_Toc361898818)

[3.7.5 Accordion 37](#_Toc361898819)

[3.7.6 Graphs 37](#_Toc361898820)

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 ABB Totalflow Android application.

1.2 References

1. ABB\_NGHLA\_Layout\_Specs\_220812.pdf
2. Intern Project – Tcox with Elaine.docx
3. Project\_Requirements\_Document.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 |
| 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 (Intern Project – Tcox with Elaine.docx) and the corresponding NGHLA Website. 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 Android Totalflow application will have method for exporting data to a file that can be shared externally.
6. Android device is only going to read the AGA-3 and AGA-7 tubes in the G4.

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 *Intern Project – Tcox with Elaine.docx*. ABB Android application is developed to enable users to browse, monitor, 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 implementation are described in subsequent sub sections.

2.1.1 Development of MCCU Android Application (*MRS1*)

MCCU 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.

2.1.3 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.

2.1.4 Design for application page development for test automation

Application page development to support multiple Android OS versions and application page testing is automated to successfully run on various supported Android devices. Look & Feel (minor differences might be there), 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.

2.1.5 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 (TRS1, TRS2)

Totalflow MCCU solution is developed as an Android application that is deployed onto the Project Wiki. 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 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.

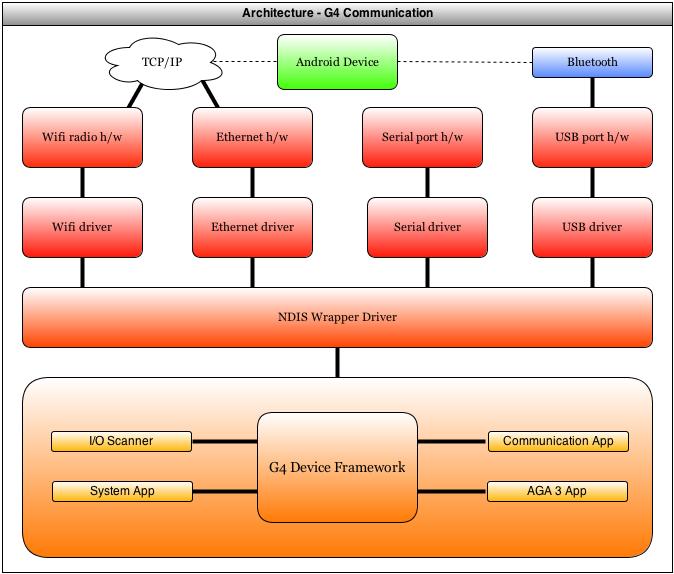


Figure 1 Solution Architecture – G4 Device

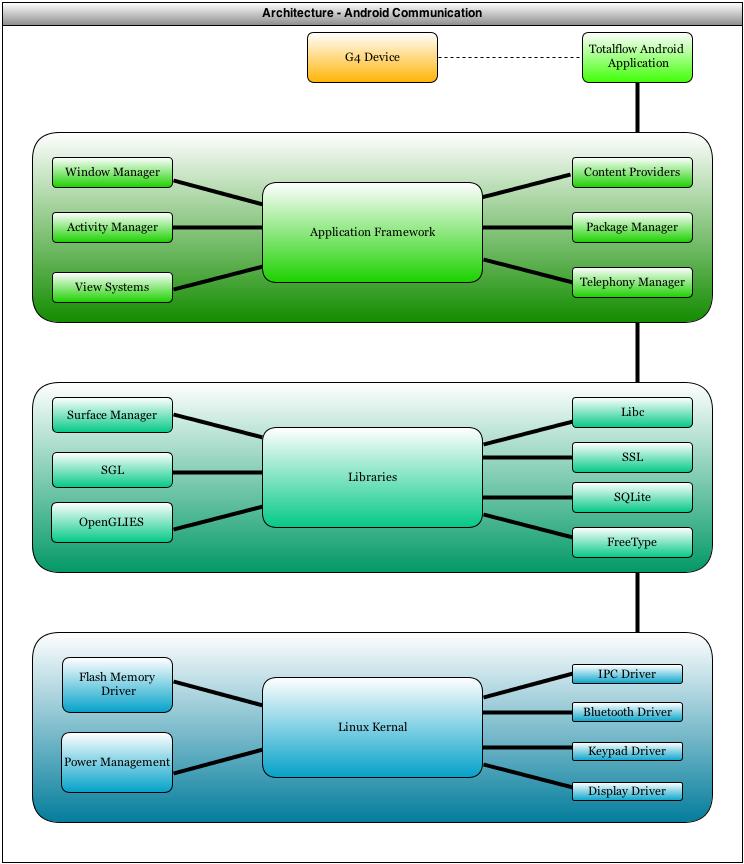


Figure 2 Solution Architecture – Android Device

## 2.4 UI Information Model (TRS4, TRS5)

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

* Connection Selection
* Login
* Sync
* Data View
* Action Bar

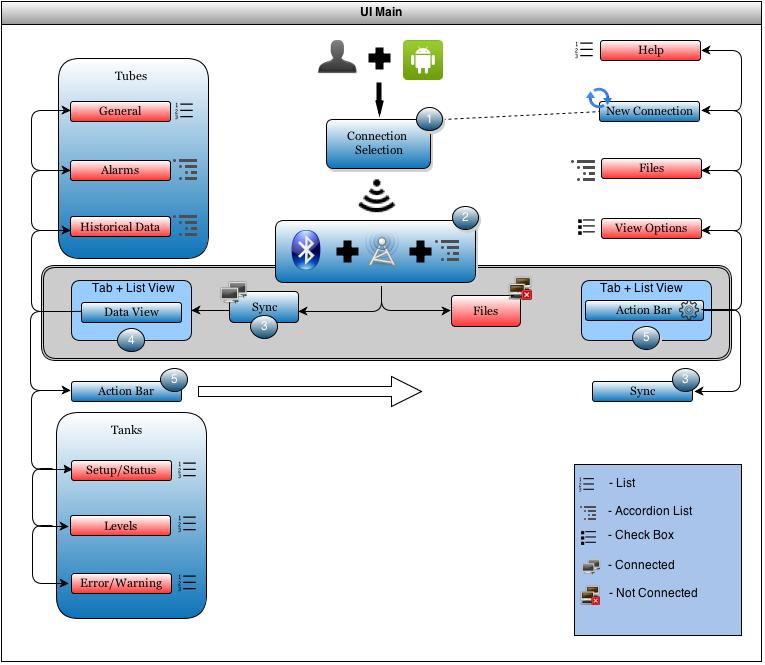


Figure 3 Login, Home Screen

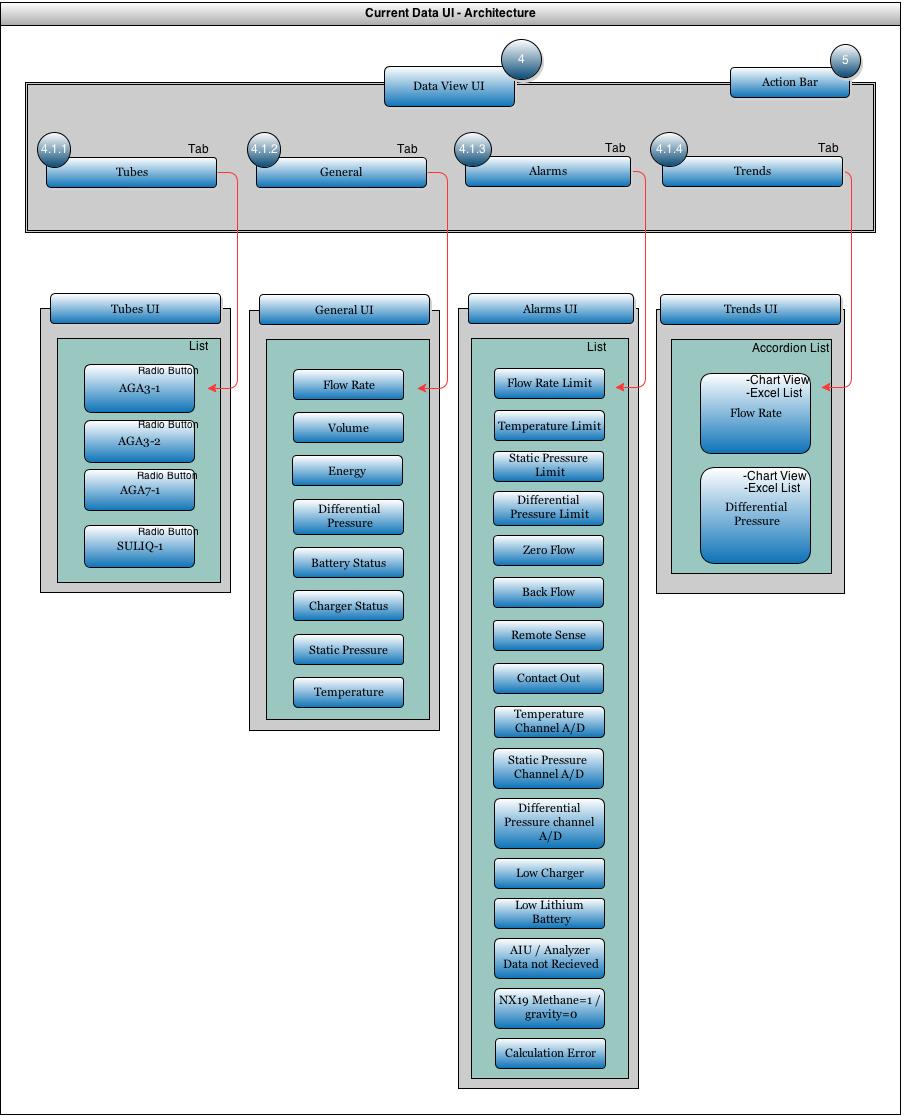


Figure 4 Data View UI

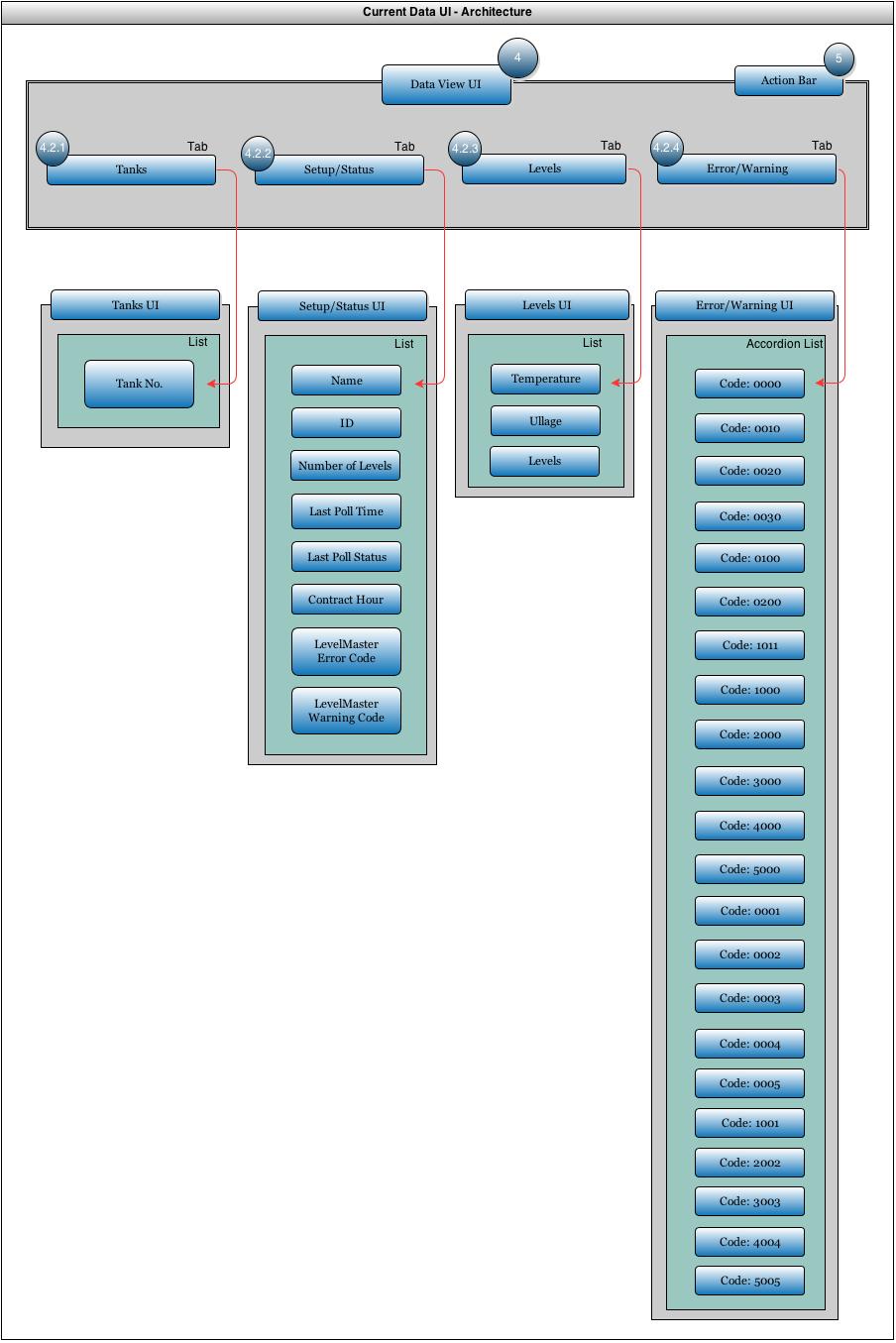


Figure 5 Data View UI (2)

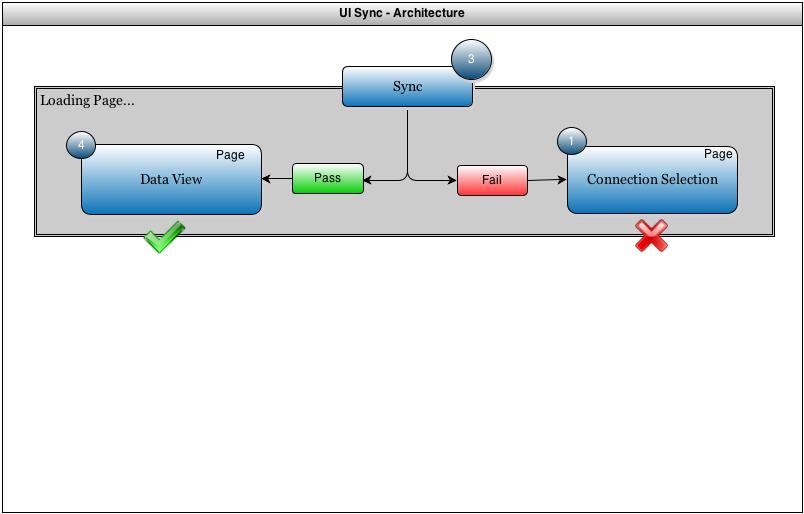


Figure 6 Sync UI (MORE DETAIL!)

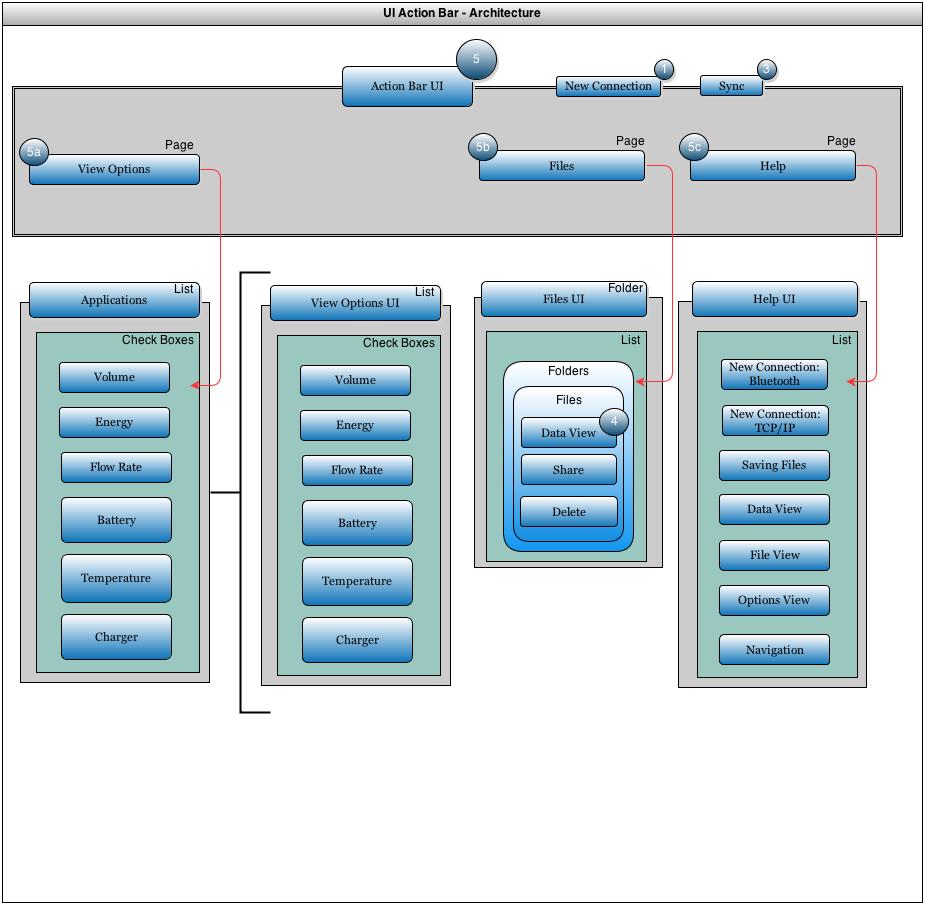


Figure 7 Action Bar UI

## 2.5 UI Screens Summary

## 2.5.1 Phase 1 Screens

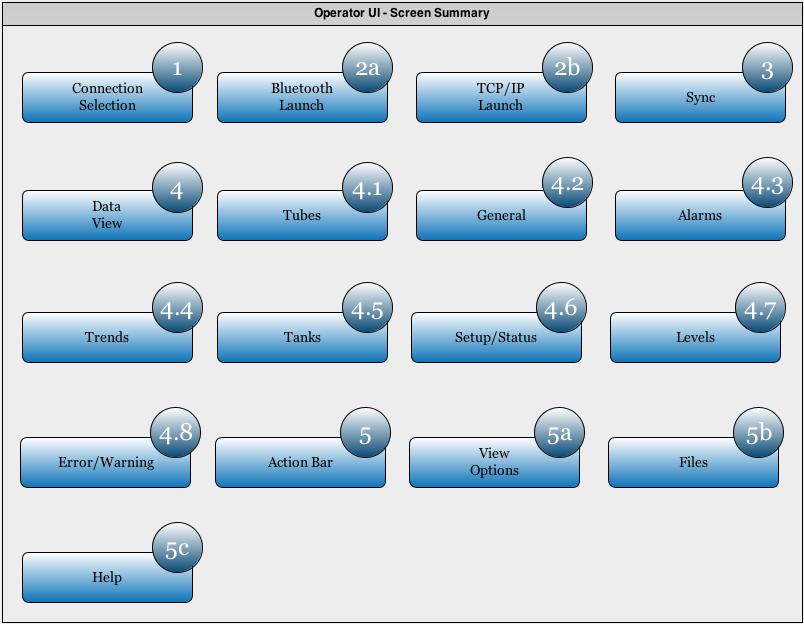


Figure 8 Remote Access UI Pages Screen Summary

3. Totalflow Android Application Design

3.1 Application Design Overview

MCCU is an application running under the context of Totalflow process which is responsible for providing similar functionality to NGHLA Web software (*TRS5*). This application is responsible for accessing different registers that contain current alarms and current/historical flow measurement (*TRS4*). The data is then stored on the Android’s external memory for accessing at a later time. The Totalflow Android 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 (*TRS1, TRS2*).

Scope of Design development work includes:

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

## 3.2 Totalflow Android Application Modules

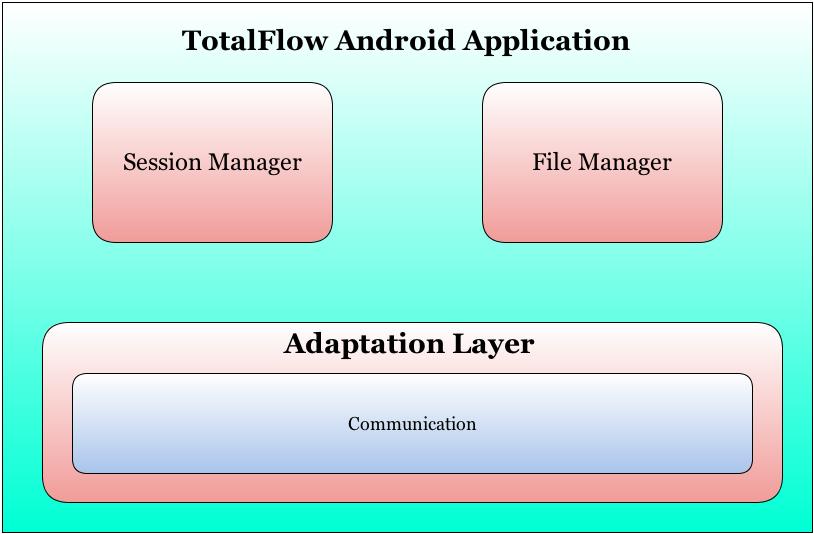


Figure 9 Totalflow Web Application Modules

Following are the modules in Totalflow Android application

* Session Manager
  + Session Manager is responsible for guiding navigation throughout application pages.
* File Manager
  + File manager is responsible for providing user access to the Android device’s external storage.
* Adaptation Layer Ask Terry (communication manager)

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

## 3.2.1 Session Manager

Upon establishing connection to a G4 device (Application Page 1), the Navigator is configured to maintain up to one user session. The user session is limited to the duration that the user chooses to navigate within the Android Application until the application is closed. 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 3.3 and 3.4.

Navigator (section 3.5) is a class containing a list that keeps track of 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 file, and then the user goes to the page to view the file data. At this point the navigator tracks the page previously visited. While the user is navigating through the application pages, the action bar menu and the data views are kept static within the Navigator. Any other navigation through the application is a linear process.

*public class Navigator {*

**navIndex indicates the current page that the user is on**

*private static int navIndex = 0;*

*private static ArrayList<String> prevPages = new ArrayList<String>();*

**Returns the previous page that the user visited**

*public static String getPrevPageName() {*

*if (navIndex == 0) {*

*return null;*

*} else {*

*decrementNav();*

*String temp = prevPages.get(navIndex);*

*prevPages.remove(navIndex);*

*return temp;*

*}*

*}*

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.

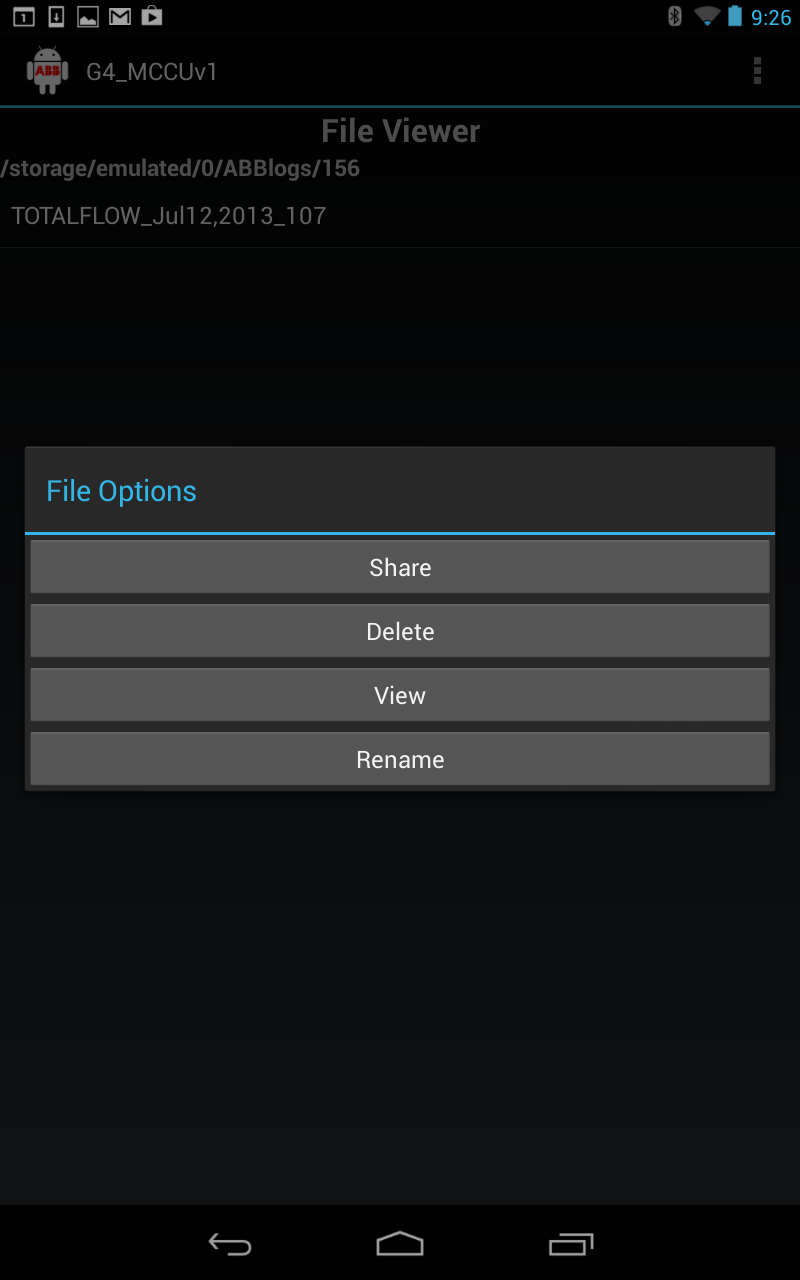
Action Bar Section/Code

## 3.2.2 File Manager

MCCU has the option of creating and saving files when the user does a long click on a file name. The user is prompted by a

MCCU creates a .CSV file to share from the Android device. The .CSV file is generated from MCCU when the user selects the “share” option. A pop-up selection box then gives the user different options to share the .CSV file. 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.

**Pop-up option for managing files:**



**Creating folder array for the user to save files:**

*public boolean onItemLongClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {*

*fileName = folderArrList.get(arg2);*

*final Dialog optionsDialog = new Dialog(context);*

*optionsDialog.setContentView(R.layout.fileviewoptionsdialog);*

*optionsDialog.setTitle("File Options");*

**Creating buttons for the different pop-up actions the user can take:**

*Button bShare = (Button) optionsDialog.findViewById(R.id.bDialogShare);*

*Button bDelete = (Button) optionsDialog.findViewById(R.id.bDialogDelete);*

*Button bView = (Button) optionsDialog.findViewById(R.id.bDialogView);*

*Button bRename = (Button) optionsDialog.findViewById(R.id.bDialogRename);*

**Creating exceptions where the user cannot save a folder:**

*optionsDialog.show();*

*bShare.setOnClickListener(new OnClickListener() {*

*public void onClick(View v) {*

*if (fileLevel.equals("")) {*

*Toast toast = Toast.makeText(context, "Cannot send entire folder, please select file.", Toast.LENGTH\_SHORT);*

*toast.setGravity(Gravity.CENTER, 0, 0);*

*toast.show();*

*} else {*

*convertBytes(readFile(direct + "/" + fileName));*

*dataIndex = 0;*

*fileNames = new ArrayList<Uri>();*

*Intent sendIntent = new Intent(Intent.ACTION\_SEND\_MULTIPLE);*

*for (int i = 1; i < allAppData.size(); i++) {*

*chosenTube = allAppData.get(i);*

*sendFile();*

*fileNames.add(Uri.parse(shareFile.getAbsolutePath()));*

*}*

*sendIntent.setType("text/csv");*

*sendIntent.putExtra(Intent.EXTRA\_STREAM, fileNames);*

*sendIntent.addFlags(Intent.FLAG\_GRANT\_READ\_URI\_PERMISSION);*

*startActivityForResult(sendIntent, 1);*

*setResult(1, sendIntent);*

*}*

*optionsDialog.dismiss();*

*}*

*});*

## 3.2.3 Totalflow Android Application Lifecycle

The Totalflow Android application pages lifecycle allows for activity functionality throughout the various pages of the application.

* **Activity Launched** calls creating and starting the process of the application. The user has selected which method of connection will be used to the G4 device. Redundancy throughout the lifecycle allows for all process to return to this point.
* **Activity Running** used in the syncing page after the process has been launched and the user has selected which method of connection will be used. 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 by navigating to the sync button from any page. If the user decides to re-sync 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 re-sync their Android device and establish a new connection at the sync page. 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 Totalflow Android 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 Totalflow Application Lifecycle

3.3 Totalflow Android Bluetooth Lifecycle (*TRS1*)

## 

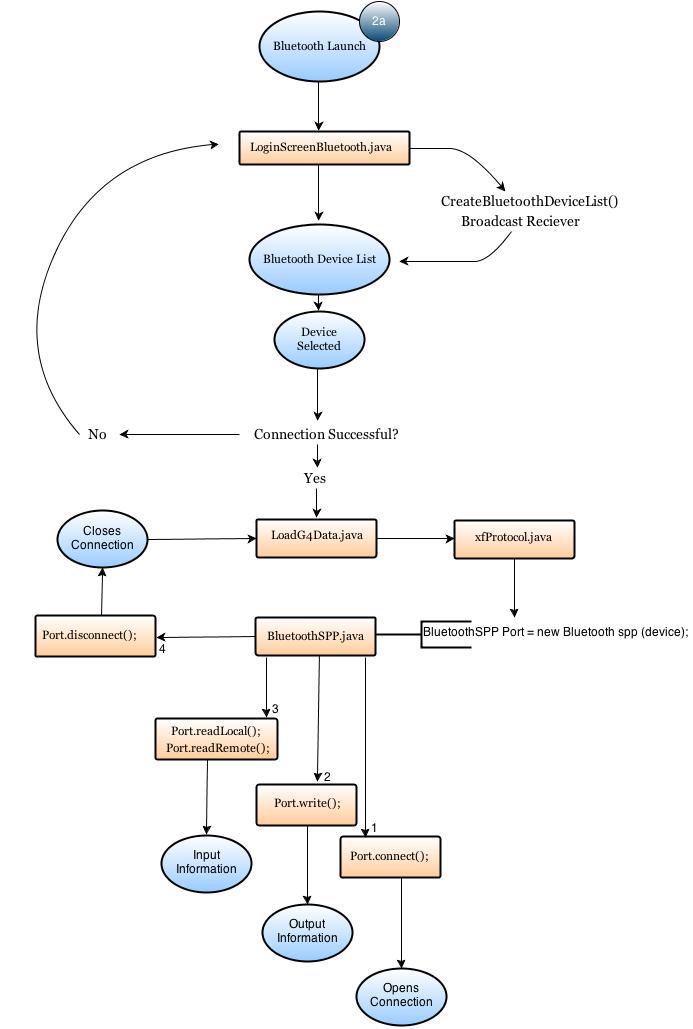


Figure 11 Bluetooth Application Lifecycle

The Android application 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 device by calling ***LoginScreenBluetooth.java*** :

*checkBTStatus();*

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

*showLoginScreen();*

*private void createBluetoothDeviceList()*

The Bluetooth device list then goes through and adds the devices and their address to a viewable list for the user to pick from. When it finds a device, the MAC address and name of device will be returned along with other requested information:

**private** **void** createBluetoothDeviceList() {

mReceiver = **new** BroadcastReceiver() {

**public** **void** onReceive(Context context, Intent intent) {

String action = intent.getAction();

**if** (BluetoothDevice.*ACTION\_FOUND*.equals(action)) {

BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.*EXTRA\_DEVICE*);

*bluetoothDevices*.add(device);

sArrBluetoothDevices.add(device.getName() + "\n" + device.getAddress());

**if** (debug) {

tvDeviceInfo.append(device.getName() + "\n" + device.getAddress() + "\n");

}

lvBTDeviceList.requestLayout();

}

}

};

After the connection list has been established, the application then jumps into ***LoadG4Data.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. ***LoadG4Data.java*** calls on functions within ***xfProtocol.java*** to access the Bluetooth Serial Port Protocol. This information requested along will be obtained after the port connects via Bluetooth. The sockets created in ***BluetoothSPP.java*** use the following methods to establish the ***Port.connect(),*** the ***Port.write(),*** ***Port.readLocal(),*** ***Port.readRemote(),*** and the ***Port.disconnect().***

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

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

**try** {

socket = device.createRfcommSocketToServiceRecord(*uuid*);

socket.connect();

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

**while**(device.getBondState()!=BluetoothDevice.*BOND\_BONDED*) {

**try** {

Thread.*sleep*(100);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

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** (Exception e) {

disconnect();

e.printStackTrace();

ret=0;

}

**return** ret;

}

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

**public** **int** ReadLocal() {

**int** totalRead=0;

**byte**[] buffer = **new** **byte**[10];

**try** {

Thread.*sleep*(1750);

totalRead= inStream.read(buffer);

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

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

returningData[i]=buffer[i];

}

}

**catch** (IOException e) {

totalRead=0;

}

**catch** (Exception e) {

totalRead=0;

e.printStackTrace();

}

**return** totalRead;

}

* **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*(200);

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[20]!=0x01 && returningData[13]==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.isConnected()) {

socket.close();

}

ret=0;

} **catch** (IOException e) {

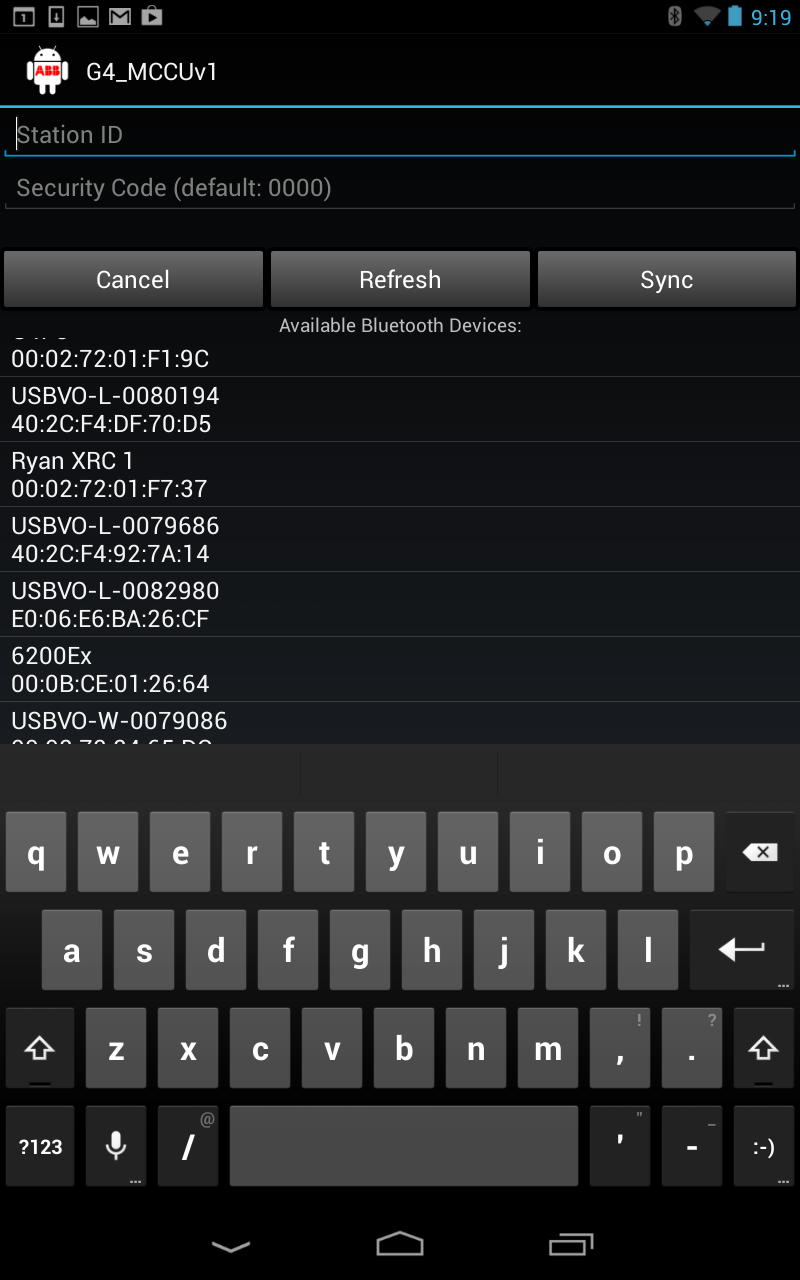
e.printStackTrace();

}

**return** ret;

}

**Screenshot of page-2a for Bluetooth log in:**



## 3.4 Totalflow Android IP Lifecycle (*TRS2*)

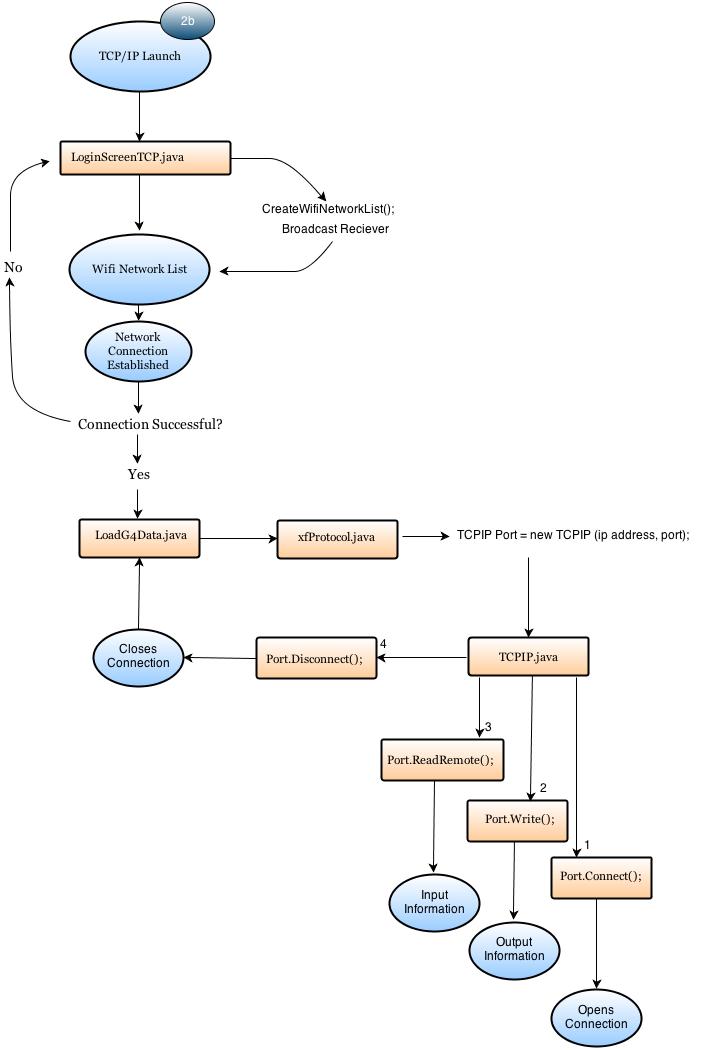


Figure 12 TCP/IP Application Lifecycle

The Android application uses Transmission Control Protocol and Internet Protocol 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 TCP/IP Connection, the Application then checks to see if the Wifi setting is turned on within the Android device by calling ***LoginScreenTCP.java***.

*checkWifiStatus();*

After the Wifi status is checked, the application turns it off then back on to refresh the list of wireless networks to connect to. This is done so by creating the Bluetooth device list:

*showLoginScreen();*

public void showLoginScreen()

The Wifi Network device list then goes through and adds the devices and their address to a viewable list for the user to pick from. When the application finds a device to be selected, the user must enter the station ID and the IP address and name of device will be returned along with other requested information. If the user cannot connect, the application will throw the error message “Unable to connect to specified network” and the user will be sent back to the login:

bSync.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View arg0) {

stationId = etStationId.getText().toString().trim();

password = etPassword.getText().toString().trim();

port = etPort.getText().toString().trim();

ipAddress = etIPaddress.getText().toString().trim();

ConnectivityManager connManager = (ConnectivityManager) getSystemService(*CONNECTIVITY\_SERVICE*);

NetworkInfo mWifi = connManager.getNetworkInfo(ConnectivityManager.*TYPE\_WIFI*);

NetworkInfo mMobile = connManager.getNetworkInfo(ConnectivityManager.*TYPE\_MOBILE*);

**if**(!tvLoginTCPInfo.getText().equals("")) {

**if**(connectToNetwork()) {

connManager = (ConnectivityManager) getSystemService(*CONNECTIVITY\_SERVICE*);

**while** (mWifi.isConnected()==**false**) {

**try** {

Thread.*sleep*(200);

} **catch** (InterruptedException e) {

}

}

} **else** {

Toast toast = Toast.*makeText*(context, "Unable to connect to specified network.",Toast.*LENGTH\_SHORT*);

toast.setGravity(Gravity.*CENTER*, 0, 0);

toast.show();

}

}

**if**(mWifi.isConnected() || (mMobile.isConnected() && tvLoginTCPInfo.getText().equals(""))) {

**if** (stationId.equals("")) {

Toast toast = Toast.*makeText*(context, "Please enter a Station ID.",Toast.*LENGTH\_SHORT*);

toast.setGravity(Gravity.*CENTER*, 0, 0);

toast.show();

} **else** **if**(ipAddress.equals("")) {

Toast toast = Toast.*makeText*(context, "Please enter an Ip Address.",Toast.*LENGTH\_SHORT*);

toast.setGravity(Gravity.*CENTER*, 0, 0);

toast.show();

} **else** {

**if** (password.equals("")) {

password = "0000";

} **if**(port.equals("")) {

port="9999";

}

unregisterReceiver(mReceiver);

Bundle basket = **new** Bundle();

basket.putString("passwordKey", password);

basket.putString("stationIdKey", stationId);

basket.putString("ipKey", ipAddress);

basket.putString("portKey", port);

basket.putInt("connectTypeKey", 1);

Intent intent = **new** Intent(LoginScreenTCP.**this**, LoadG4Data.**class**);

intent.putExtras(basket);

startActivity(intent);

finish();

}

}

}

});

After the connection list has been established, the application then jumps into ***LoadG4Data.java*** to begin the requests needed for data sets from different registers. ***LoadG4Data.java*** calls on functions within ***xfProtocol.java*** to access the Network Internet Protocol. Every Network connection requires a “client” and “host” to establish the sockets. This information requested along will be obtained after the port connects via Bluetooth. The sockets created in ***BluetoothSPP.java*** use the following methods to establish the ***Port.connect(),*** the ***Port.write(),*** ***Port.readLocal(),*** ***Port.readRemote(),*** and the ***Port.disconnect().***

***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) {

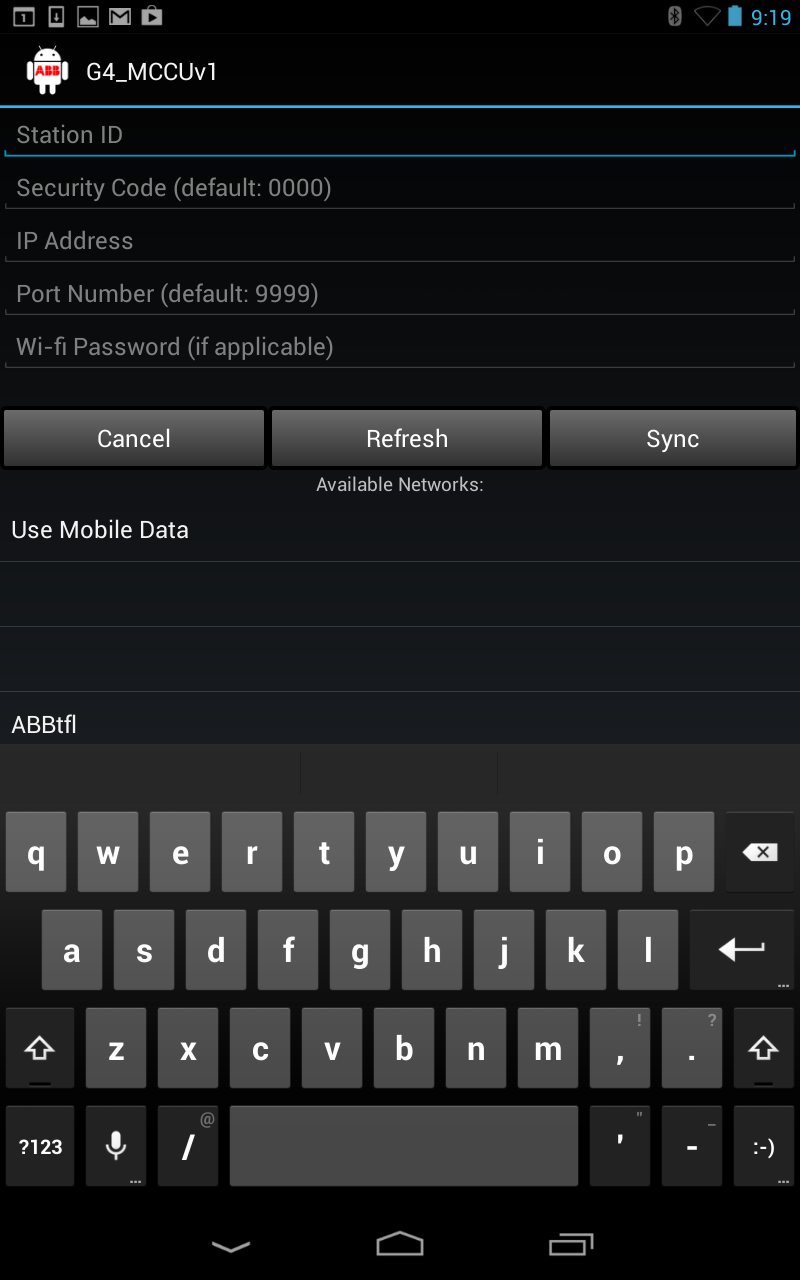
e.printStackTrace();

}

**return** ret;

}

**Screenshot of page-2b for TCP/IP connection:**



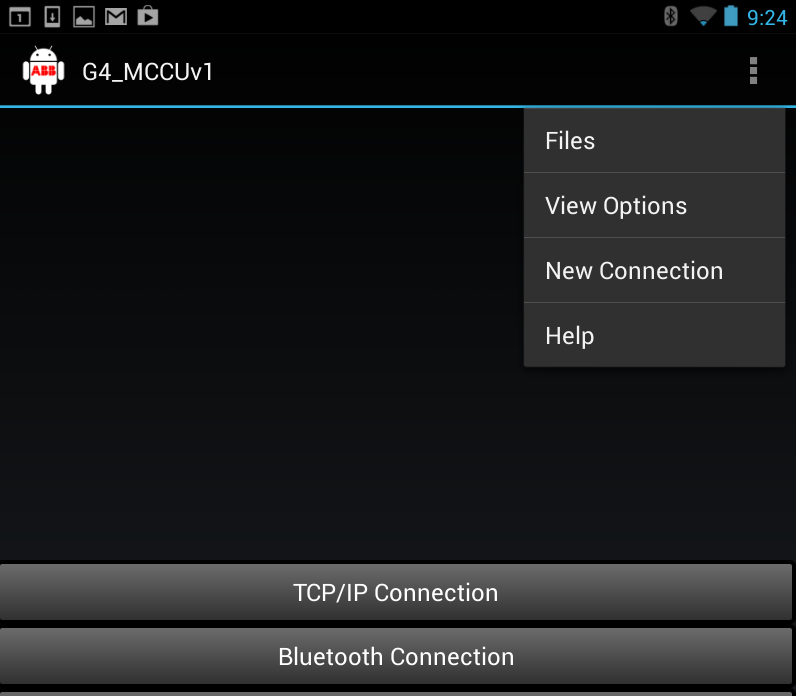
## 3.5 Design Aspects for Application Performance

The Android devices this application will be used on have limited resources in terms of memory and CPU. In order to achieve the time for individual page loading to around 1 second and comparable to the time taken by similar screen on MCCU, following parameters need to be considered and adjusted. The other aspect that needs to be considered is the hardware capability of the Android device used to run the application.

The **Navigator** (section 3.2.1) always keeps the user at square one, allowing for every page to open seamlessly with no hanging pages. This may not be necessary on every page, but it is more efficient to keep track of all pages in order to optimize user navigation throughout the application.

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 is located in the upper right corner of the screen for new Android devices. The Action Bar is a physical touch button located below the screen for older Android devices. The Action Bar is represented by three dots inside of a square. The ease of use adds to the applications efficiency.

**Screenshot of upper right corner where the Action Bar Tab is located:**



## 3.5.1 Page Size and Number of Objects

Number of objects and size of all objects on a web page have a direct impact on the response time for that application page. To improve the performance of any application page, the number of objects needs to be minimized. The techniques for these will be handled by having two separate sizes of objects to be used for Android phones and Android tablets. In order to reduce the size of the objects used on any application page there are some techniques like compressing the content, reducing the image sizes to the desired resolution.

## 3.5.2 Number of Threads

The Android application has one thread dedicated to the user interface. The Android OS forces all network communications onto a separate thread. A new thread is created at the beginning of each data transaction; after transaction is completed the thread is then finished. If the transaction is canceled then an error is forced into the protocol causing it to finish the thread early in order to maximize performance. There are two additional threads being used for display purposes during loading for page-1 and page-3.

**Upon application startup an image temporarily displayed for the duration of 2 seconds:**



## 3.5.3 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 that will get called when the user presses the “back” button to navigate to the previous page. This functionality is shown as follows:

**should be called when user goes to a new page. The page they were at previously should be entered as @prevPage:**

*public static void addPrevPage(String prevPage) {*

*prevPages.add(prevPage);*

*incrementNav();*

*}*

*private static void decrementNav() {*

*navIndex--;*

*}*

*private static void incrementNav() {*

*navIndex++;*

*}*

*public static ArrayList<String> getArray() {*

*return prevPages;*

*}*

Files generated from the data transaction are stored externally on the Android device to the SD memory card. The files are saved to a dedicated folder *ABBlogs*. Sub folders can then be created within *ABBlogs* to help differentiate the location, device, etc. Preferences are saved to internal storage and cannot be directly accessed by the user, they can only be deleted when deleting the entire application or through an Android application manager. The access of folders in external memory is shown below, created in the LoadG4Data class:

*final Dialog newFolderDialog = new Dialog(context);*

*newFolderDialog.setContentView(R.layout.filerenamedialog);*

*newFolderDialog.setTitle("Create New Folder");*

*Button bConfirm = (Button) newFolderDialog.findViewById(R.id.bRenameDialogYes);*

*Button bCancel = (Button) newFolderDialog.findViewById(R.id.bRenameDialogNo);*

*final EditText etNewName = (EditText) newFolderDialog.findViewById(R.id.etNewFileName);*

*etNewName.setText("");*

*etNewName.setHint("New Folder Name");*

*newFolderDialog.show();*

*bConfirm.setOnClickListener(new OnClickListener() {*

*public void onClick(View arg0) {*

*String newFolderName = etNewName.getText().toString().trim();*

*if(newFolderName.equals("")) {*

*Toast toast = Toast.makeText(context, "No name entered.",Toast.LENGTH\_SHORT);*

*toast.setGravity(Gravity.CENTER, 0, 0);*

*toast.show();*

*} else if(newFolderName.contains("<")||newFolderName.contains(">")||newFolderName.contains(":")||newFolderName.contains("\"")|| newFolderName.contains("\\")||newFolderName.contains("/")||newFolderName.contains("|")||newFolderName.contains("?")||*

*newFolderName.contains("\*")) {*

*Toast toast = Toast.makeText(context, "Folder name cannot contain special characters:\n< > : \" \\ / | ? \*",Toast.LENGTH\_SHORT);*

*toast.setGravity(Gravity.CENTER, 0, 0);*

*toast.show();*

*} else if(newFolderName.charAt(0)=='.') {*

*Toast toast = Toast.makeText(context, "Folder name cannot begin with '.'",Toast.LENGTH\_SHORT);*

*toast.setGravity(Gravity.CENTER, 0, 0);*

*toast.show();*

*} else if(folderArrList.contains(newFolderName))*

*Toast toast = Toast.makeText(context, "Folder already exists.",Toast.LENGTH\_SHORT);*

*toast.setGravity(Gravity.CENTER, 0, 0);*

*toast.show();*

*} else {*

*selectedFolderName=newFolderName;*

*tvFolderName.setText(selectedFolderName);*

*}*

*}*

*});*

## 3.5.4 Recovery from crash/hang

If the Android application crashes due to some problem/bug and causes Android MCCU to stop in that case, the user must re-open the Android application and start again at page-1. If the application crashes during a sync or file storage, the data is terminated. If the application crashes during options view, the user preferences are not saved. If the application crashes during SYNC, the port is left hanging and the user must manually reset the G4 device. If the Android device does not get a response from the G4 device, it will throw a timeout error.

## 3.5.5 Use of MVC Architecture

Model-View-Controller design pattern helps in separating the business logic, presentation logic and navigation logic. Models are responsible for encapsulating the application data. The Views render response to the user with the help of the model object. Controllers are responsible for receiving the request from the user and calling the back-end services.

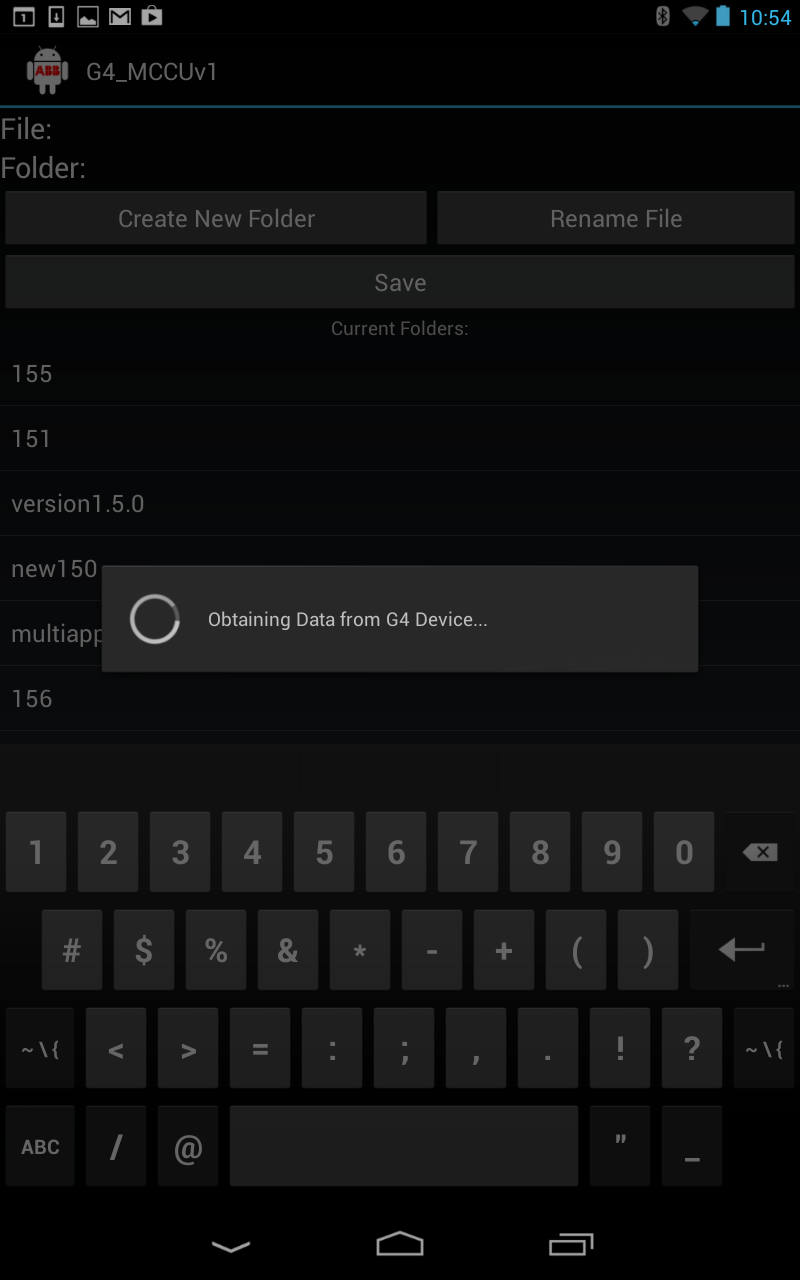
The MVC design is used specifically when the user navigates to the data trends to view the charts for data of the past 14 days. The user is also able to select which sets of data are available to view from their View Options Page (Application Page 5a).

## 3.5.6 Indicative/Responsive UI

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

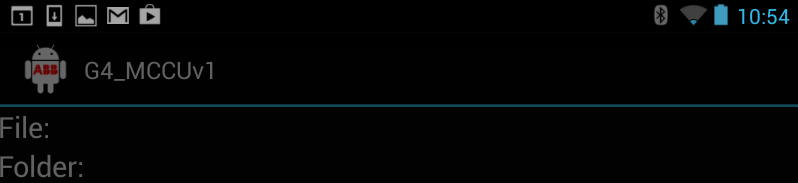
After the user inputs their information for connecting to a device via Bluetooth or IP, the application will display the dialogue spinner to let the user know that their action is being process while the software makes all the necessary socket connections on the back end.

**Screenshot example for “spinner” used when loading data:**



In the case that the Android device will reset the Bluetooth or Wifi status from off/on, then a suitable progress 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, then the status will be displayed at the top bar of the screen.

**Screenshot example with Bluetooth and Wifi status at top right:**



## 3.6 Registers (TRS4)

The Android application only uses the read functionality when talking to the G4 device. Only the predefined registers are accessed by the Android device when the user connects to the G4 device (TRS4). The data is then accessible by the user to either store into the Android device’s external memory to be shared or to be displayed in lists and charts from the device.

## 3.6.1 Active Alarms

When reading active alarms from the G4 device, the user will be able to see any unacknowledged alarms. This will not allow the user to acknowledge 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 as unacknowledged.

The Android application is enhanced to provide the list of current Active Alarms based on the specific Tube containing the List of Registers. The current Tubes being measured are: AGA-3, AGA-7, and Liquids. This list of registers is the set of data points which are associated with a Station. 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 Tube request and the Alarm request are as follows:

**AGA-3**

*xfRequest tubeName = new xfRequest(aga3,5,0, 1, 65); numStrings++;*

*xfRequest alarms = new xfRequest(aga3, 9, 0, 1, int32.sizeof()); numInt32++;*

*registerList[index] = alarms;*

All Active Alarms are requested in following format:

(AppName, Array, Index Start, Index Total, Size);

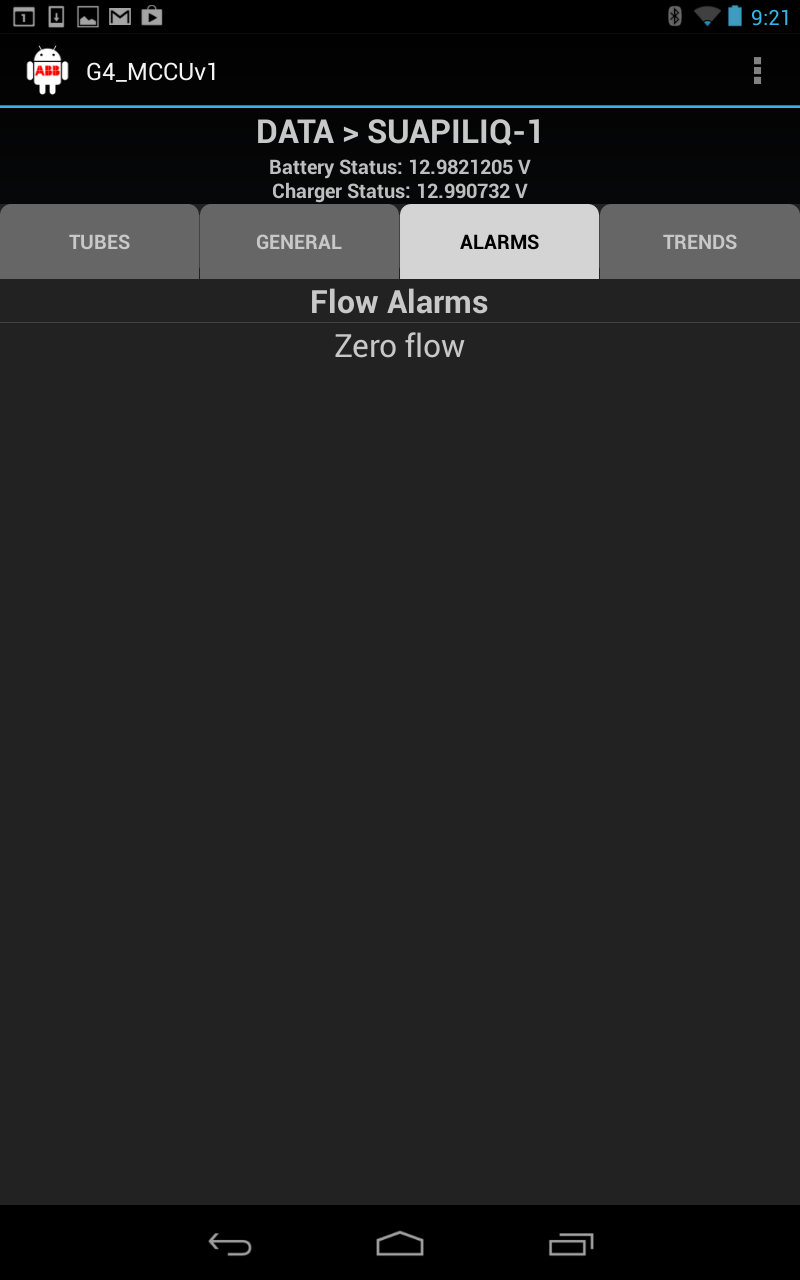
A class is then created before entering the connection protocol to allow for the data coming back from the Alarm register request. The registerData class contains the Array that will be used to hold the number of bytes for all the desired data types:

*registerData = new RegisterData(numStrings, numFloats, numInt32, numBytes);*

*memory= new byte[numFloats\*Float.SIZE/8 + numInt32\*int32.sizeof() + numBytes\*Byte.SIZE/8 + numStrings\*65];*

After the Array has been created, the connection protocol for either Bluetooth or TCP/IP will begin (section 3.3 and 3.4), and the register request for alarms will be carried out.

**Screenshot example of Alarm Tab Data:**



## 3.6.2 Alarm Log

## The Android application 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.

## 3.6.3 General Descriptions

When reading general descriptions from the G4 device, the user will be able to see and select the tube name for which set of data they would like to display. After the user has synced with the G4 device, they will obtain a list of tubes that can be selected for viewing. The user must re-sync from the Android application to further update their data taken from the G4 device. For each time frame of measurement, the user will be able to see volume, flow rate, energy, energy rate, static pressure, differential pressure, temperature, pulse, and mass.

The Android application is enhanced to provide the list of the flow measurements based on the specific Tube containing the List of Registers. The current Tubes being measured are: AGA-3, AGA-7, and Liquids. This list of registers is the set of data points which are associated with a Station. A register request corresponding to the Tubes Flow Measurements are 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 Tube request and the Alarm request are as follows*:

**AGA-3 and Liquids**

*xfRequest tubeName = new xfRequest(aga3,5,0, 1, 65); numStrings++;*

*xfRequest tubeName = new xfRequest(liq,5,0, 1, 65); numStrings++; numReg++;*

*registerList[index]=tubeName; index++;*

All Active Alarms are requested in following format:

(AppName, Array, Index Start, Index Total, Size);

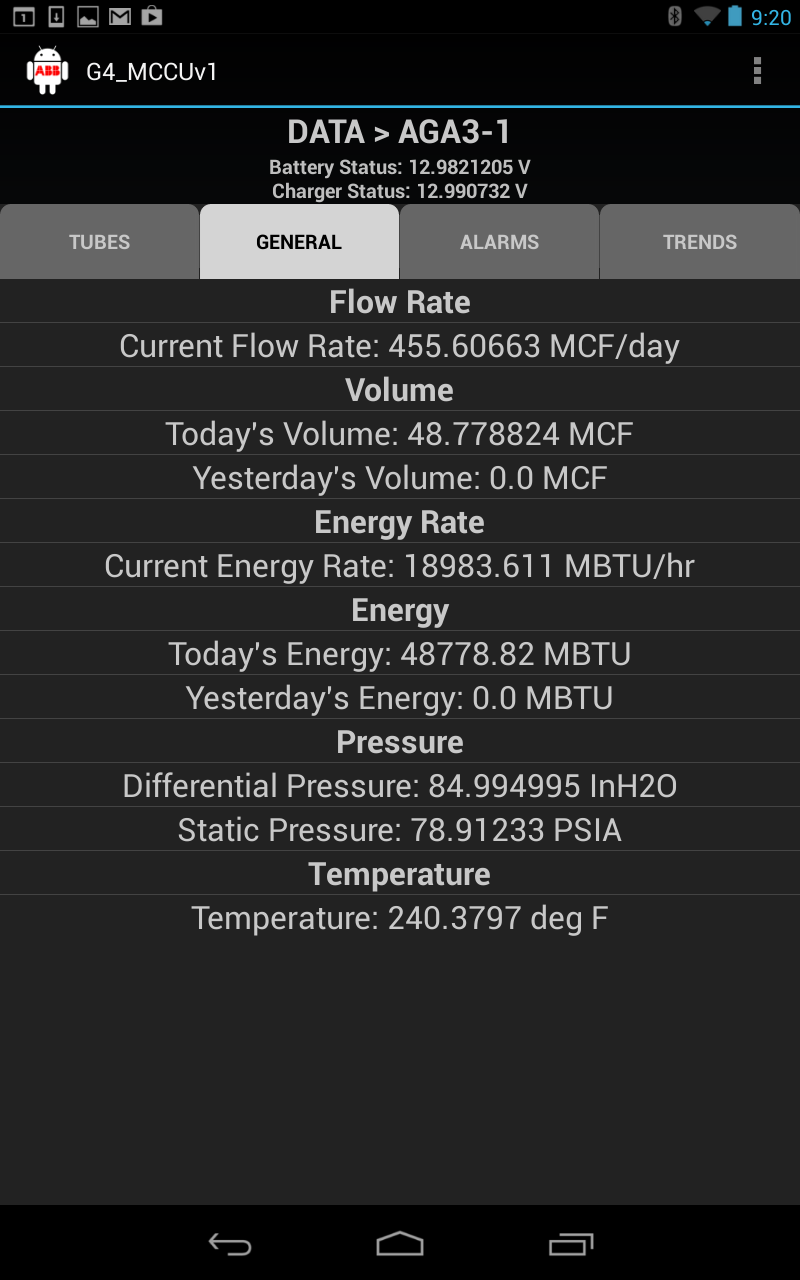
A class is then created before entering the connection protocol to allow for the data coming back from the Tube name register request. The registerData class contains the Array that will be used to hold the number of bytes for all the desired data types:

*registerData = new RegisterData(numStrings, numFloats, numInt32, numBytes);*

*memory= new byte[numFloats\*Float.SIZE/8 + numInt32\*int32.sizeof() + numBytes\*Byte.SIZE/8 + numStrings\*65];*

After the Array has been created, the connection protocol for either Bluetooth or TCP/IP will begin (section 3.3 and 3.4), and the register request for alarms will be carried out.

**Screenshot example of General Tab Data:**



## 3.6.4 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. After the user has synced with the G4 device, they will obtain a “snapshot” of all these data points up to the time they connected by syncing with the G4 device. The user must re-sync from the Android application to further update their data taken from the G4 device. For each time frame of measurement, the user will be able to see volume, flow rate, energy, energy rate, static pressure, differential pressure, temperature, pulse, and mass.

The Android application is enhanced to provide the list of the flow measurements based on the specific Tube containing the List of Registers. The current Tubes being measured are: AGA-3, AGA-7, and Liquids. This list of registers is the set of data points which are associated with a Station. A register request corresponding to the Tubes Flow Measurements are 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 Tube request and the Alarm request are as follows*:

**AGA-3, AGA-7, and Liquids**

*xfRequest tubeName = new xfRequest(aga3,5,0, 1, 65); numStrings++;*

*xfRequest todayV = new xfRequest(aga3, 7, 22, 1, floatSize); numFloats++; numReg++;*

*xfRequest yesterdayV = new xfRequest(aga3, 7, 23, 1, floatSize); numFloats++; numReg++;*

*xfRequest currentER = new xfRequest(aga3, 3, 73, 1, floatSize); numFloats++; numReg++;*

*xfRequest todayE = new xfRequest(aga3,3,71, 1, floatSize); numFloats++; numReg++;*

*xfRequest yesterdayE = new xfRequest(aga3,3,72, 1, floatSize); numFloats++; numReg++;*

*xfRequest currentFR = new xfRequest(aga3, 7, 19, 1, floatSize); numFloats++; numReg++;*

*xfRequest temperature = new xfRequest(aga3, 3, 3, 1, floatSize); numFloats++; numReg++;*

*xfRequest DP = new xfRequest(aga3, 7, 0, 1, floatSize); numFloats++; numReg++;*

*xfRequest SP = new xfRequest(aga3, 3, 0, 1, floatSize); numFloats++; numReg++;*

*xfRequest volume14 = new xfRequest(aga3, 225, 0, 14, floatSize); numFloats += 14; numReg++;*

*xfRequest pressure14 = new xfRequest(aga3, 221, 0, 14, floatSize); numFloats += 14; numReg++;*

*xfRequest pulse = new xfRequest(aga7su,7,0, 1, floatSize); numFloats++; numReg++;*

*xfRequest accumulatedMass = new xfRequest(liq,36,9, 1, floatSize); numFloats++; numReg++;*

*registerList[index] = currentFR; index++;*

*registerList[index] = todayV; index++;*

*registerList[index] = yesterdayV; index++;*

*registerList[index] = currentER; index++;*

*registerList[index] = todayE; index++;*

*registerList[index] = yesterdayE; index++;*

*registerList[index] = DP; index++;*

*registerList[index] = SP; index++;*

*registerList[index] = temperature; index++;*

*registerList[index] = volume14; index++;*

*registerList[index] = pressure14; index++;*

*registerList[index] = pulse; index++;*

*registerList[index] = todayMass; index++;*

*registerList[index] = yesterdayMass; index++;*

*registerList[index] = accumulatedMass; index++;*

All Active Alarms are requested in following format:

(AppName, Array, Index Start, Index Total, Size);

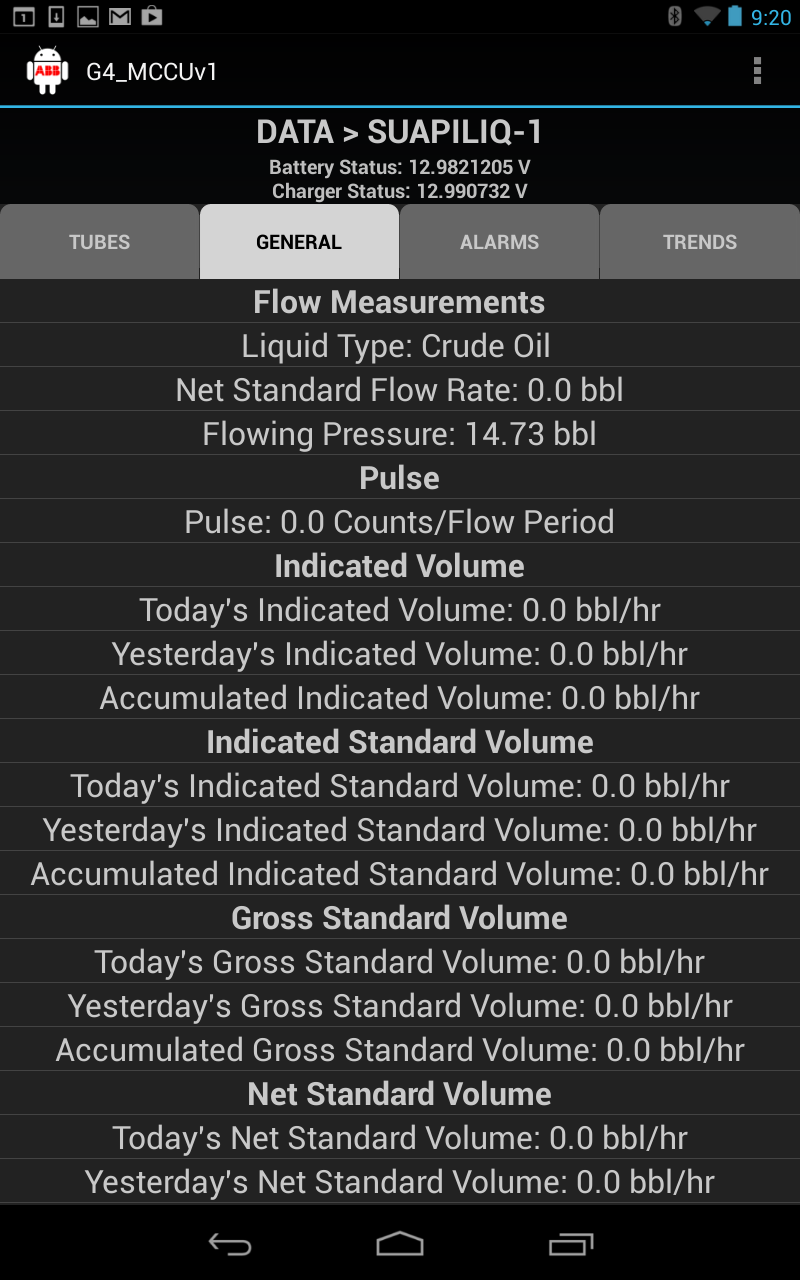
A class is then created before entering the connection protocol to allow for the data coming back from the Flow Measurement register request. The registerData class contains the Array that will be used to hold the number of bytes for all the desired data types:

*registerData = new RegisterData(numStrings, numFloats, numInt32, numBytes);*

*memory= new byte[numFloats\*Float.SIZE/8 + numInt32\*int32.sizeof() + numBytes\*Byte.SIZE/8 + numStrings\*65];*

After the Array has been created, the connection protocol for either Bluetooth or TCP/IP will begin (section 3.3 and 3.4), and the register request for alarms will be carried out.

**Screenshot example for Flow Measurement Data collected:**



## 3.6.5 Units

When reading flow measurement from the G4 device, the user will be able to see all data with corresponding units of measurement. After the user has synced with the G4 device, they will obtain a “snapshot” of all these data points up to the time they connected by syncing with the G4 device. The user must re-sync from the Android application to further update their data taken from the G4 device. For each time frame of measurement, the user will be able to see units corresponding to volume, flow rate, energy, energy rate, static pressure, differential pressure, temperature, pulse, and mass.

The Android application is enhanced to provide the list of the flow measurements with corresponding units based on the specific Tube containing the List of Registers. The current Tubes being measured are: AGA-3, AGA-7, and Liquids. This list of registers is the set of data points which are associated with a Station. A register request corresponding to the Tubes Flow Measurements are 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 Tube request and the Alarm request are as follows*:

*xfRequest frUnit= new xfRequest(aga3su,11,20,1,65); numStrings++; numReg++;*

*xfRequest vUnit= new xfRequest(aga3su,11,18,1,65); numStrings++; numReg++;*

*xfRequest erUnit= new xfRequest(aga3su,11,30,1,65); numStrings++; numReg++;*

*xfRequest eUnit= new xfRequest(aga3su,11,28,1,65); numStrings++; numReg++;*

*xfRequest dpUnit= new xfRequest(aga3su,11,3,1,65); numStrings++; numReg++;*

*xfRequest spUnit= new xfRequest(aga3su,11,2,1,65); numStrings++; numReg++;*

*xfRequest tUnit= new xfRequest(aga3su,11,5,1,65); numStrings++; numReg++;*

*xfRequest mUnit= new xfRequest(aga3su,11,13,1,65); numStrings++; numReg++;*

*xfRequest ivUnit= new xfRequest(aga3su,11,23,1,65); numStrings++; numReg++;*

*registerList[index]=frUnit; index++;*

*registerList[index]=vUnit; index++;*

*registerList[index]=erUnit; index++;*

*registerList[index]=eUnit; index++;*

*registerList[index]=dpUnit; index++;*

*registerList[index]=spUnit; index++;*

*registerList[index]=tUnit; index++;*

*registerList[index]=mUnit; index++;*

*registerList[index]=ivUnit; index++;*

## 3.6.6 Level Master

The level Master Application is enhanced to add a new register which is used to provide the total capacity of the tank. The total capacity is computed by adding the height of the various sections defined in the tank. This register will return capacity in feet and inches.

Level Master application is also enhanced to add register for Unit of Temperature. Unit for volume is always Bbls. These values for US devices is F and Bbls.

The MCCU Application uses register request to gather the information from the level master application. There are two different register requests made to the level master. The first request is to get the number of tanks and the second request is to get the data within the tank requested:

**The first request:**

*private byte getNumTankRegister(int tank) {*

*numStrings=0;*

*numBytes=0;*

*numFloats=0;*

*numInt32=0;*

*numInt16=0;*

*numUInt8=0;*

*numInt8=0;*

Adding the requests to a list:

*int numReg=0;*

*xfRequest numTanksReg = new xfRequest(tank,107,0, 1, int16.sizeof()); numInt16++; numReg++;*

*xfRequest[] registerList = new xfRequest[numReg];*

*int index = 0;*

*registerList[index]=numTanksReg; index++;*

To hold the returning data:

*registerData = new RegisterData(numStrings, numBytes, numFloats, numInt32, numInt16,numUInt8, numInt8);*

*db2Protocol = new xfProtocol(code, stationID, registerList, registerData, device,ipAddress, port, connectType, false);*

*db2Protocol.ReadRegister();*

*registerData=db2Protocol.getRegisterData();*

*return db2Protocol.getStatus();*

*}*

**The second request to gather all the register information for the tank chosen by the user:**

***private byte getTankRegister(int tank, int numTanks) {***

***int floatSize = Float.SIZE / 8;***

***int numReg=0;***

***byte status=0;***

***int nameIndex=0, idIndex=2, levelsIndex=1, pollTimeIndex=0, pollStatusIndex=1, contractHourIndex=0, errorIndex=0, warningIndex=1;***

***int tempIndex=2, ullageIndex=5, level1InchesIndex=0, level2InchesIndex=1, level1VolumeIndex=3, level2VolumeIndex=4;***

***for(int j=0; j<numTanks; j++) {***

***if (status == 0) {***

***numReg=0;***

***numStrings=0;***

***numBytes=0;***

***numFloats=0;***

***numInt32=0;***

***numInt16=0;***

***numUInt8=0;***

***numInt8=0;***

***xfRequest tankName = new xfRequest(tank,117,nameIndex, 1, 65); numStrings++; numReg++; nameIndex+=1;***

***xfRequest pollTime = new xfRequest(tank,106,pollTimeIndex, 1, 65); numStrings++; numReg++; pollTimeIndex+=1;***

***xfRequest temperature = new xfRequest(tank,103,tempIndex, 1, floatSize); numFloats++; numReg++; tempIndex+=6;***

***xfRequest ullage = new xfRequest(tank,103,ullageIndex, 1, floatSize); numFloats++; numReg++; ullageIndex+=6;***

***xfRequest level1Inches = new xfRequest(tank,103,level1InchesIndex, 1, floatSize); numFloats++; numReg++; level1InchesIndex+=6;***

***xfRequest level1Volume = new xfRequest(tank,103,level1VolumeIndex, 1, floatSize); numFloats++; numReg++; level1VolumeIndex+=6;***

***xfRequest level2Inches = new xfRequest(tank,103,level2InchesIndex, 1, floatSize); numFloats++; numReg++; level2InchesIndex+=6;***

***xfRequest level2Volume = new xfRequest(tank,103,level2VolumeIndex, 1, floatSize); numFloats++; numReg++; level2VolumeIndex+=6;***

***xfRequest errorCodes = new xfRequest(tank,102,errorIndex, 1, int16.sizeof()); numInt16++; numReg++; errorIndex+=4;***

***xfRequest warningCodes = new xfRequest(tank,102,warningIndex, 1, int16.sizeof()); numInt16++; numReg++; warningIndex+=4;***

***xfRequest tankId = new xfRequest(tank,100,idIndex, 1, uint8.sizeof()); numUInt8++; numReg++; idIndex+=3;***

***xfRequest numLevels = new xfRequest(tank,100,levelsIndex, 1, uint8.sizeof()); numUInt8++; numReg++; levelsIndex+=3;***

***xfRequest pollStatus = new xfRequest(tank,105,pollStatusIndex, 1, int8.sizeof()); numInt8++; numReg++; pollStatusIndex+=3;***

***xfRequest contractHour = new xfRequest(tank,115,contractHourIndex, 1, int8.sizeof()); numInt8++; numReg++; contractHourIndex+=2;***

***xfRequest[] registerList = new xfRequest[numReg];***

***int index=0;***

***registerList[index]=tankName; index++;***

***registerList[index]=pollTime; index++;***

***registerList[index]=temperature; index++;***

***registerList[index]=ullage; index++;***

***registerList[index]=level1Inches; index++;***

***registerList[index]=level1Volume; index++;***

***registerList[index]=level2Inches; index++;***

***registerList[index]=level2Volume; index++;***

***registerList[index]=errorCodes; index++;***

***registerList[index]=warningCodes; index++;***

***registerList[index]=tankId; index++;***

***registerList[index]=numLevels; index++;***

***registerList[index]=pollStatus; index++;***

***registerList[index]=contractHour; index++;***

***registerData = new RegisterData(numStrings, numBytes, numFloats, numInt32, numInt16,numUInt8, numInt8); // This will hold the all the returning data***

***db2Protocol = new xfProtocol(code, stationID, registerList, registerData, device,ipAddress, port, connectType, false);***

***db2Protocol.ReadRegister();***

***registerData=db2Protocol.getRegisterData();***

***fillBuffer(AppView.LEVELMASTER);***

***status=db2Protocol.getStatus();***

***}***

***}***

***return status;***

***}***

Screen shots for tank selection

## 3.6.7 Warning and Error Codes

When using the Android MCCU application there are cases where the application may not follow a request made by the user. The application is designed to notify the user when there is an error throughout the user session for the application. When reading data from the LevelMaster application, there are warnings and error codes that need to be displayed by the Android MCCU application as well. For the instance of an error popping up, the MCCU application runs through the following cases to check and then display to the user if an error is flagged.

*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.";*

*}*

*}*

## 3.7 Templates

Templates are made to maximize the code reuse. These template files capture the UI elements which are used at multiple places. Appropriate styling information for UI elements is maintained in the RES folder files. This way, based on the screen requirement, one can simply use the *color.xml*, *dimens.xml*, *strings.xml*, and *styles.xml* files to create new pages for the application.

UI Implementation is done as per the screens in the MCCU.apk. Look and feel, UI components, layout and placing of UI controls, are done as per MCCU.apk. Implementation of screens will follow the below principles which were followed for designing the screens:-

* Boolean value

These types of values are displayed using ‘Checkbox’ control.

* Enumeration Value

If values to be shown are 3 or less than it then they are represented as a radio checkbox control. If there are more than 3 possible values then a combo box shall be used.

* String Data

For non-editable field, labels are used and for editable fields, textbox is used. The maximum limit of characters for text filed is 200.

* Floating Point

These values are displayed with their precision part and the editable fields should allow the user to enter the precision parts also easily with controls like ‘UI Spinner’.

**Fonts**

In general, Arial Font, default theme size is used as the ‘font’ across the screens to display various text. On all screens for non-value labels font is non-bold and all values are bold.

**Error Scenario Handling**

In case there is an error while generating the application pages, requesting data, saving data, etc, then an error page is displayed to the user. This error page displays brief error description message along with possible recovery step (if required). This error page helps the user to be aware of any possible error and then recover from it (see section 3.6.7).

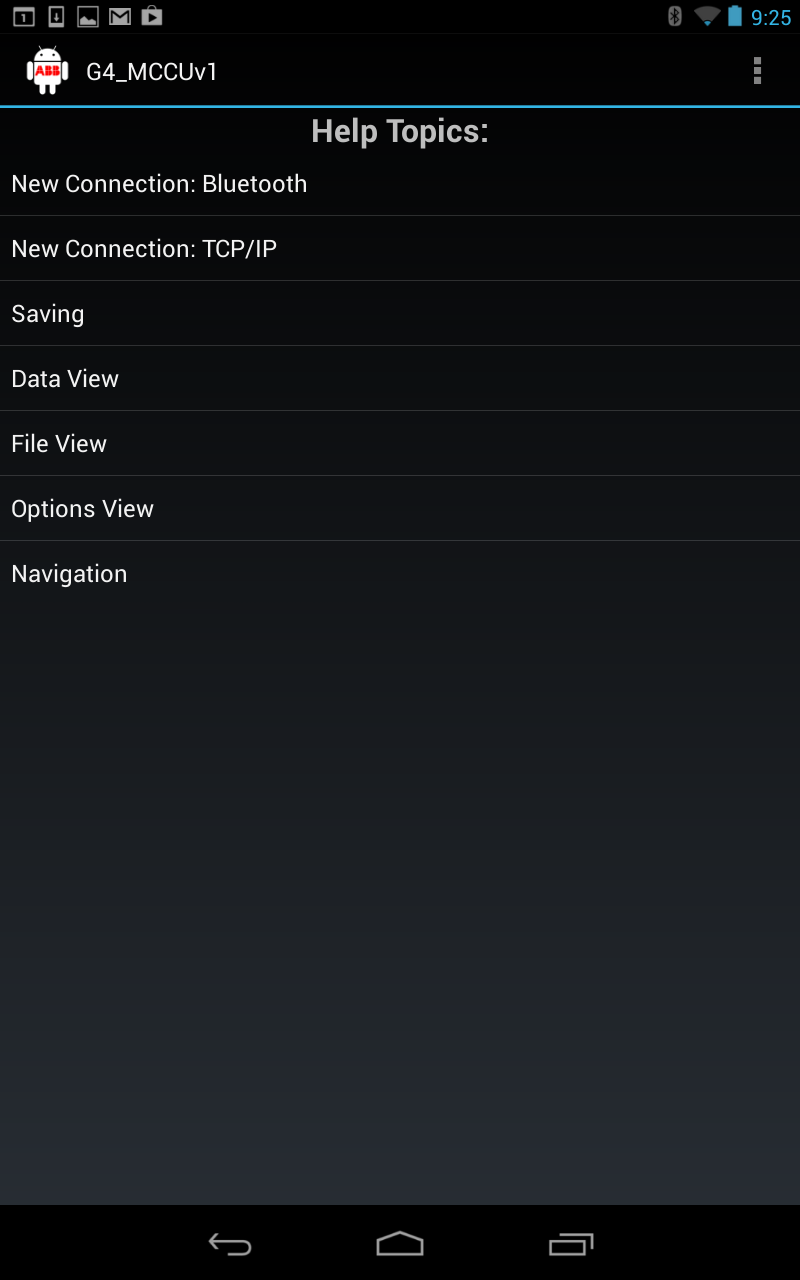
**No Data from Register**

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 3.6.7.

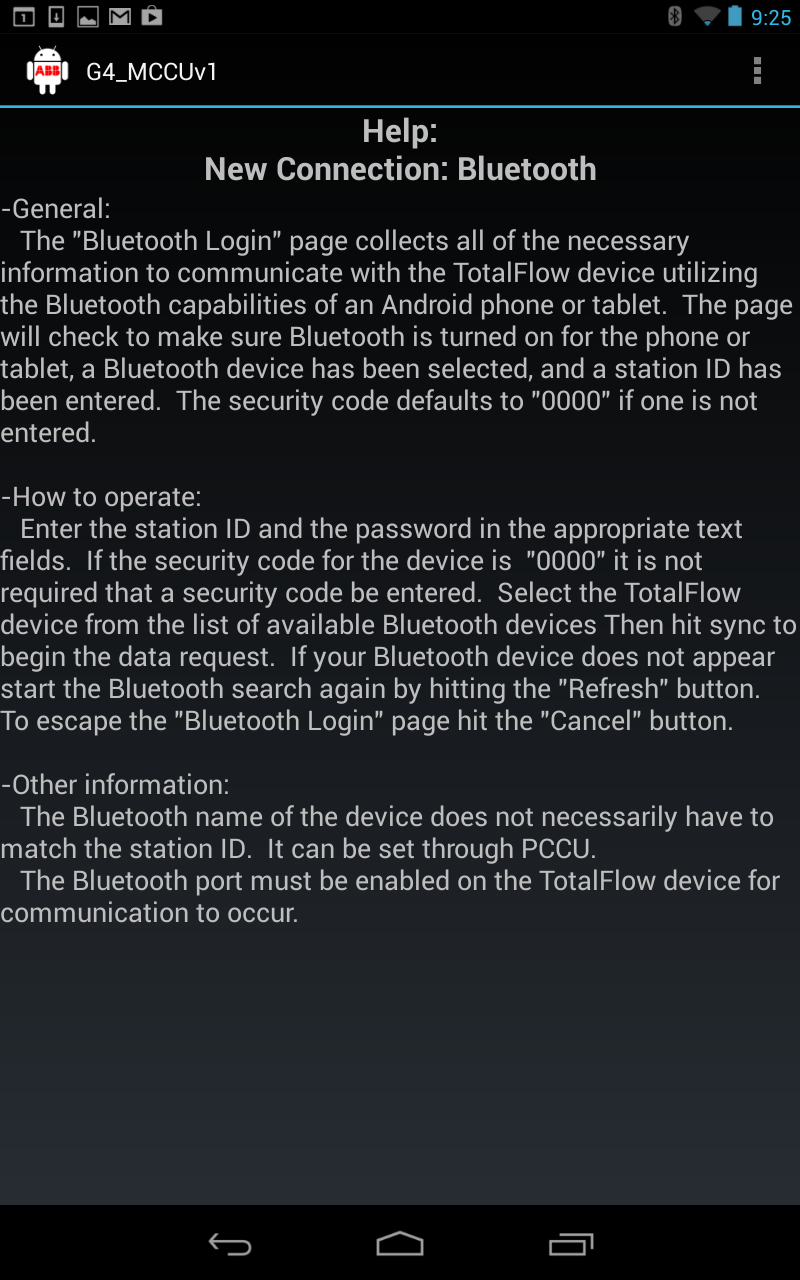
**Help**

In the current phase of the implementation, the help options are available within the action bar. The help page will allow the user to navigate information on topics regarding connection for Bluetooth/IP, saving files, data view, file view, options view, and navigation.

Screenshot from Help menu:



Screenshot example of selecting a Help option:



## 3.7.1 Template Guideline

* Capture the basic layout – This way the same XML template are placed inside an accordion or it can be placed inside a Tab. So, we define the template to contain one logical piece of correlated data. This data is placed in various types of container like Tab, Accordion. Templates defined there do not specify any accordion. It’s only at runtime they pick one XML page and then put the XML content in one part of another XML page. This part can be a Tab, Accordion or a simple div or span.
* Define the basic UI components and serve as placeholder for the runtime data.
* Styles information will be applied as classes. All such classes are maintained in RES files.
* For multiple repeating elements, there is only one entry in the template. This single entry has the unique Base-Name Id which is updated at runtime for the newly added entries when the colors.xml and layoutview.xml files are read.
* Template files are read at different point of time. The xml content of the template is given as input to other template/pages

## 3.7.2 Res Folder

Styling information for the user interface part is maintained as separate files. CSS is used to maintain the style information like background image, font-style, size etc. All such information is maintained as ‘Classes’ which are applied as required at multiple places. This also ensures that styling information is not duplicated.

## 3.7.3 Action Bar

The action bar 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 is populated with various tabs to allow the user for navigating throughout the application. Remote Access pages have the menu which will provide options to move across various pages and perform various actions. This menu also contains information for no connection and establishing a new connection. This menu is present across all pages (will conform to the menu options as displayed in Section 2.4) and will conform to the fonts and colors called from the application’s RES folder.

## 3.7.4 Tab Navigation

User can choose to browse across various screen sections using the different Tabs. This tab section is present upper thirds section of the screen. Every tab contains the name that describes the content on screen which gets visible 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: Arial
* Font Size: Theme Default
* Width : Theme Default
* Height: Theme Default
* Font Color: Light Grey (#c8c8c8)
* Background Color: Very Dark Grey (#666666)

1. Selected

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

## 3.7.5 Accordion

The Data View page contains the Trends Tab with a lot of data that is organized in accordions. The Accordion is a view control that allows you to provide multiple panes and display them when user clicks on them. It is like having several [panels](http://www.asp.net/ajaxlibrary/AjaxControlToolkitSampleSite/CollapsiblePanel/CollapsiblePanel.aspx) where every accordion can be expanded on user click. Every accordion represents a logical grouping of the data.

Accordions make it easy to display large number of data in a limited screen space. Accordions will conform to following specification:-

1. Accordion Section Content

* Font Style: Arial
* Font Size: Default Theme
* Font Color: Very Light Grey (#d4d4d4)
* Background Color: Very Dark Grey (#222222)

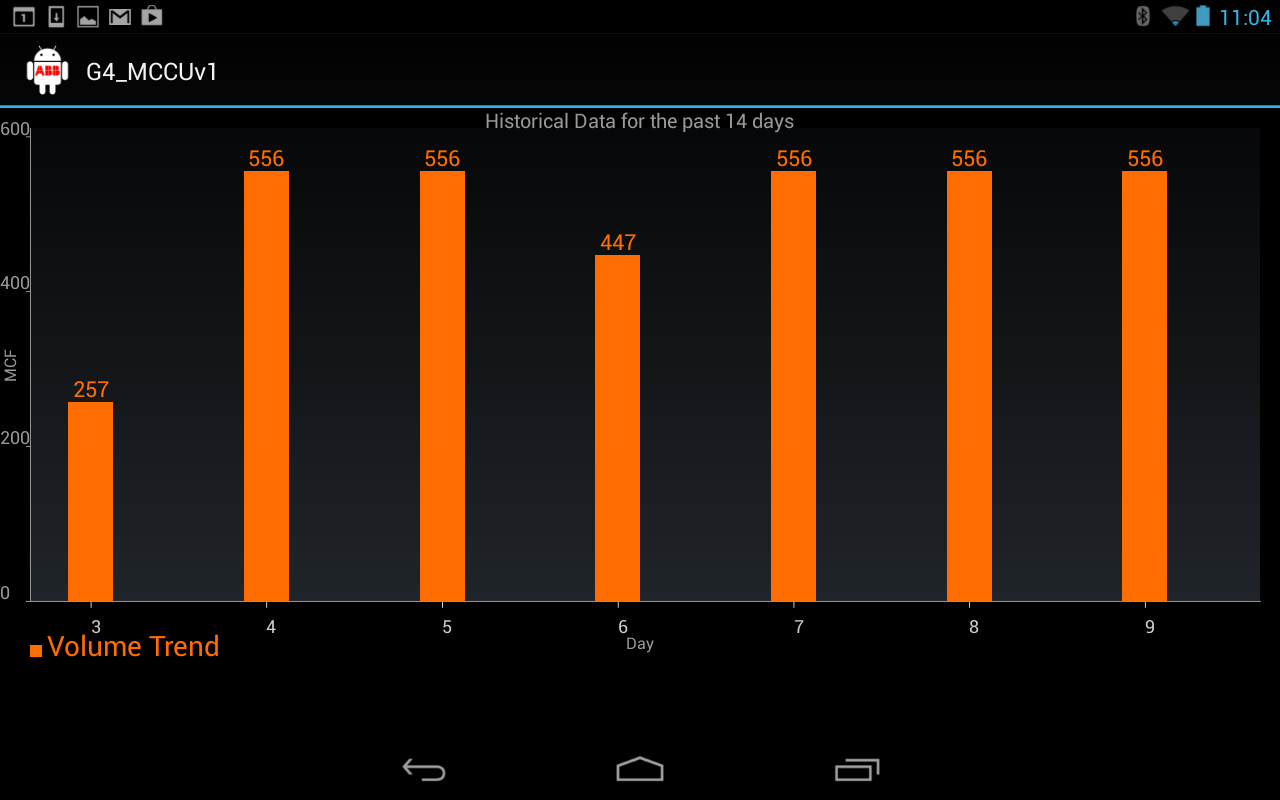
## 3.7.6 Graphs

There will be graphs used to look at the Trends’ data to visualize the various parameters and monitor them over a period of time. With the help of the graph, observed parameters can be graphically depicted and compared. 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. Along with the 2 axis, there will also be legends to describe multiple parameters in case there are multiple parameters that are being used in the graph for comparison with other parameters.

Graphs are of ‘Bar’ type where value of a parameter in one axis will be drawn against the value on the other axis.

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

**Screenshot for an example of historical data:**



## User Editable Fields Validation

jQuery based validation framework is used to define and handle the client side web page validations.

jQuery framework defines validation rules and error messages which can be used as part of the validation process.

Following are the key elements used in this framework:-

1. **Method**: A validation method implements the logic to validate an element, like an email method that checks for the right format of an text input's value. A set of standard methods is available, and it is also easy to write our own custom method.
2. R**ule**: A validation rule associates an element with a validation method, like "validate input with name "primary-mail" with methods "required" and "email".

So, this framework gives the flexibility to use the already defined and implemented validation for various commonly used fields. Below is the list of such built-in validation rules:-

1. **Required**: The typical and probably most used validation rule.
2. **Minlength & Maxlength**: Staple Length validation rules
3. **Rangelength**: Similar to Min/Maxlength; allows for selected item counts as well.
4. **Min & Max & Range**: Same as the "length" rules above only for numeric values.
5. **Email**: A very robust email validation rule
6. **Url**: Ensures the input is a well formatted url.
7. **Date**: Verifies the input can be parsed into a date.
8. **Number**: Ensures that the input is numeric.
9. **Digits**: Ensures only 0-9 characters (no commas, decimal points, etc.)
10. **Accept**: A file extension validation rule.
11. ***EqualTo***: Ensures that the values of 2 elements are the same

There may happen a situation where we find that the built in rules just don't cover everything we need.  When such an occurrence comes upon us, we have the ability to extend the Validation Plug-in.  Custom validation rules are, ultimately, methods that are called and return a boolean value; true if the test succeeds and false if the test fails.  After the methods are created, they can be wired up to the jQuery validation component and used like any other rule.

After we have implemented the new function, we then attach it to the jQuery Validation plug-in.  To do this, we call the validator object's addMethod() function.  This addMethod() function takes 3 parameters; a label for the rule, the function that contains the test, and the default message to display when the test fails.  Example:

$.validator.addMethod("onlyText", myNewValidationMethod, "Enter only alpha characters ( a-z ).");

Now that the function has been added, we can use the rule, called "onlyText", just like any of the built in rules.

Validation rules should come into affect when the user completes the editing of the field and tries to move it’s focus on the other controls on the page. In case of any validation error, user should be made aware of it at that time only.