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 6 |

Contents

[NOTE 1](#_Toc359828007)

[1. Introduction 2](#_Toc359828008)

[1.1 Intended Audience 2](#_Toc359828009)

[1.2 References 2](#_Toc359828010)

[1.3 Acronyms and Abbreviations 2](#_Toc359828011)

[1.4 Assumptions 2](#_Toc359828012)

[2. System Design 3](#_Toc359828013)

[2.1 System Overview 3](#_Toc359828014)

[2.1.1 Development of Totalflow Android Application 3](#_Toc359828015)

[2.1.2 Development of Remote Access App pages 3](#_Toc359828016)

[2.1.3 Development of Adaptation layer 3](#_Toc359828017)

[2.1.4 Design for application page development for test automation 4](#_Toc359828018)

[2.1.5 Totalflow.apk (Flash) Changes 4](#_Toc359828019)

[2.2 Deployment Diagram 4](#_Toc359828020)

[2.3 Solution Architecture 4](#_Toc359828021)

[2.4 UI Information Model 6](#_Toc359828022)

[2.5 UI Screens Summary 10](#_Toc359828023)

[2.5.1 Phase 1 Screens 10](#_Toc359828024)

[3 Totalflow Android Application Design 10](#_Toc359828025)

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. 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

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 |
| RBAC | Role Based Access Control |
| SSL | Secure Socket Layer |
| SPP | Serial Port Protocol |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| 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. ABB Android application is developed to enable users to browse, monitor, and share various trends and statistical values of the Totalflow device.

Scope of Remote Access development work includes:

1. Development of User Interface
2. Development of Totalflow Android application.
3. Development of Adaptation layer in which application uses Bluetooth to communicate with G4 device.
4. Design for Application page development for multiple Android devices and test automation.
5. Design for the Application Register/File Access methods (User Management, Alarm, Events and other flash changes).

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

2.1.1 Development of Totalflow Android Application

Application code is integrated into 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) within the existing registers and the trends are hosted on the flash file system. The existing RBAC is not used for this communication.

2.1.2 Development of Remote Access App pages

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

2.1.3 Development of Adaptation layer

The Adaptation layer abstracts the use of Serial Port Profiles (SPP), Transmission Control Protocol (TCP/IP), and Radio frequency communication (RFCOMM). If at any point of time the Totalflow native calls change or get modified then the Android application is not impacted. If we decide to change the Android application API or use a different mobile device only this adaptation layer needs to be modified and no change would be needed in the underlying Totalflow native calls.

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.apk (Flash) Changes

* 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 event log.

Details of these changes are mentioned in Chapter .

## 

## 2.2 Deployment Diagram



## 2.3 Solution Architecture

ABB Totalflow Bluetooth/TCP solution is developed as a set of data pages and is deployed on abb.com (running on Android device). 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 pages 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.

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

Figure 1 Solution Architecture – G4 Device

Figure 2 Solution Architecture – Android Device

## 2.4 UI Information Model

The Totalflow Android application UI can be divided into two sets of information pages, Current Data and Settings and these are further divided into various information pages.

* Current Data
* Settings

Figure 3 Login, Home Screen

Figure 4 Current data UI, Historical data UI, Alarms, and General

Figure 5 Connect Options UI, Files UI, and View Options UI

## 2.5 UI Screens Summary

## 2.5.1 Phase 1 Screens

Figure 6 Remote Access UI Pages

3. Totalflow Android Application Design

Totalflow Android application is an application running under the context of Totalflow process which is responsible for providing similar functionality to PCCU software. This application is responsible for accessing different registers that contain current alarms and current/historical flow measurement. 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.

## 3.1 Totalflow Android Application Modules

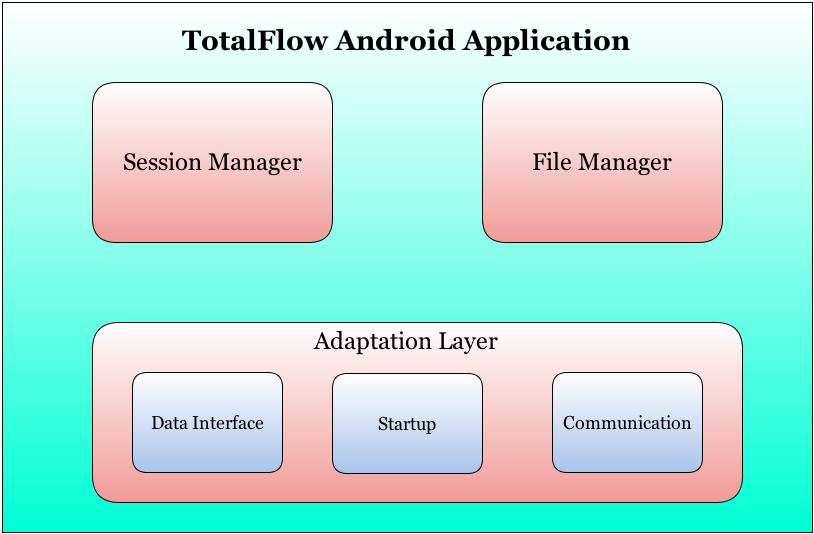


Figure 7 Totalflow Web Application Modules

Following are the modules in Totalflow Android application

* Session Manager
  + Session Manager is responsible for creation and validation of user sessions.
* File Manager
  + File manager is responsible for providing user access to historical files.
* Adaptation Layer

This layer acts as a broker between web server and Totalflow application framework and vice-versa. It is responsible for communicating with other Totalflow applications to get and change the data related to various application. It provides interfaces to the flash for the following functionalities:

* Starting the Android application
* Starting the Remote connection
* Storing the various data in the phone’s external memory
* Switching between communication protocols.

## 3.2 Totalflow Android Application Lifecycle

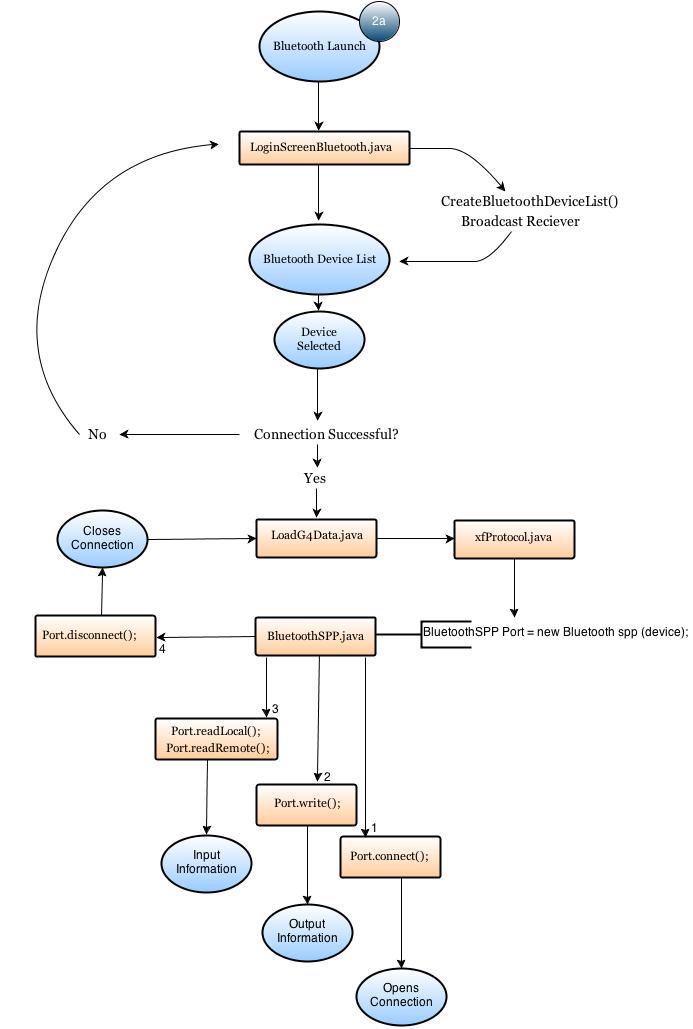
The Totalflow Android application pages lifecycle allows for activity functionality throughout the various pages of the application.

* **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.



## 3.3 Totalflow Android Bluetooth 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 in the SPP. ***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) {

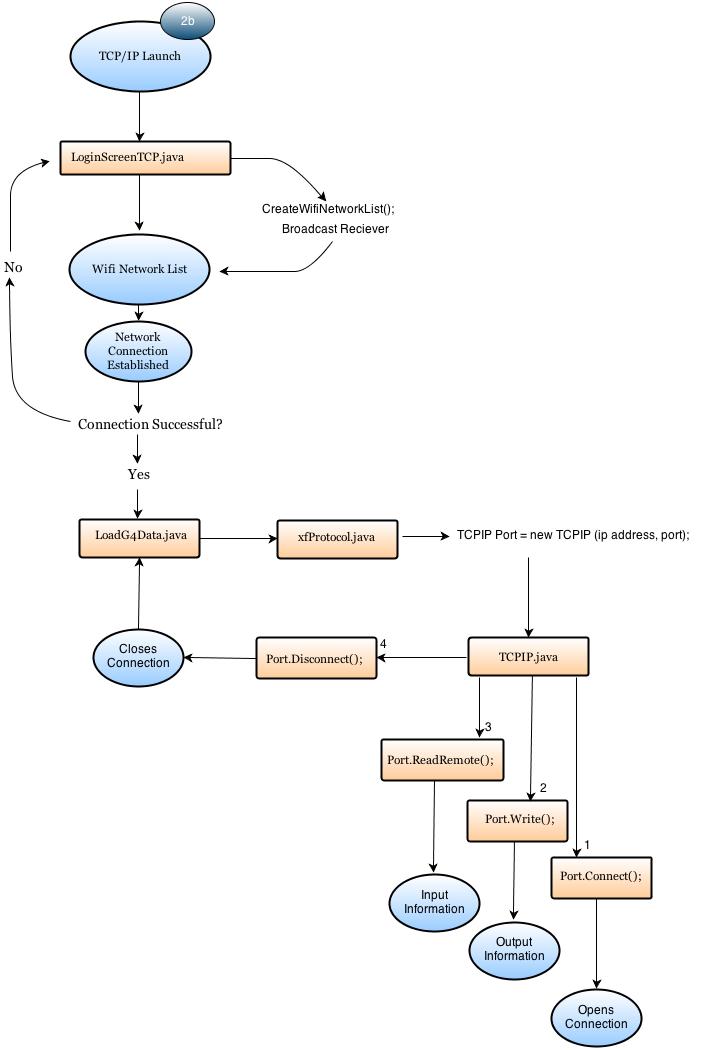
e.printStackTrace();

}

**return** ret;

}

## 3.4 Totalflow Android IP 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;

}

## 3.5 Design Aspects for Application Performance

JEFF

The Android devices this application will be used on have limited resources in terms of memory and CPU. The design of the application and memory usage shall keep the CPU usage below 85%. 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.

## 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 to 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 threads being used for display purposes during loading for page-1 and page-3.

JEFF

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:

Jeff’s Code

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:

Jeff’s Code

## 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 Session Manager Design

JEFF

Session Manager is configured to maintain up to 4 simultaneous user sessions. A cache is maintained to keep track of active user sessions. For every successful authentication, a new entry is made in this cache. If a request comes for login at the time when there are already 4 active users, in that case request is refused and the user is shown an error page asking him to try login after some time as the system has reached its allowed active user limit. If any new user logs in and if there is no session available then the session manager also checks to see if there is a session which is not used for more than 15 minutes and it clears that session and creates a new session.

If the user chooses to extend the user session then the session creation time is updated with the timestamp when the session extension request was received. If the session manager cannot extend the session for some reason then the web server return an error message to the browser indicating error.

After successful login, if the user clears the browser cache and cookies and if user now clicks on any link on the page then this request would not have the session information. In that case the web server would show an error page suggesting that the session has expired along with the link to login again.

In case the login feature is disabled, user will be redirected to the Operator UI home screen as long as the maximum session limit is not reached. In such a scenario, if user clicks the ‘Logout’ link then the current session will be terminated (along with removing all the cookie information) and user will be shown a logout confirmation page.

There is a possibility that target browser can allow user to open tabs in the same browser. In one scenario, user is browsing web pages and (s)he opens any link in page a new tab(using mouse right click) then this will result in reuse of the original session cookie. In another scenario, if Home URL for the web page is opened in a new tab then it will result in the creation of a new session. This new session information will override any existing session in the tabs for that browser window. So, effectively all the tabs in the browser window will then use the latest session information for making the request. All the old session will still be valid at the device which will eventually get expired after the session timeout.

Cookie will be used to maintain two session related information :-

* User Name
* Session-Id

For the above two information, two cookie (uname & sessionId) will be created which will be passed by browser in every requests to server after a successful login of NextGen local access web page.

User Name cookie will be used to display the user name in every web page. Session-Id cookie will be checked in server end to verify if the current request is having a valid session identifier or not. This session check needs to be part of the every page loading and display.

## 3.5.6 CVS File Generator

The purpose of the XML file is to provide flexibility to the web pages to be designed and implemented without having to know how many stations are configured, how many applications are associated with a station and what registers are used by an application. This information is stored in the XML file and the XML file is generated based on the current configuration of the device. Any time any configuration is changed and sent using the PCCU application, a new XML file is generated based on the updated configuration. This XML file is read by the web pages and based on it the web pages know which registers to read to show various Stations and various applications associated with a particular station. This XML generation is part of the PCCU application code,

* any time a new application is added or removed,
* any time a new IO point is added or removed,
* a new station added or removed,
* association/disassociation of some application with some station,
* number of tanks changed

In all other cases it does not require generation of the new XML file.

## 3.5.6 CVS File Generator

## 3.5.6 CVS File Generator

## 3.5.7 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 freezed/blocked condition or state.

In case the update on screen is going to take some time (more than 2 sec) then a suitable progress indicator is shown to the user that reflects the ongoing processing by the web page in the backend. Once the page gets loaded, then this progress indication will go away.

## 3.1 Totalflow Android Application Modules

## 3.1 Totalflow Android Application Modules

## 3.1 Totalflow Android Application Modules

## 3.1 Totalflow Android Application Modules

## 3.1 Totalflow Android Application Modules

## 4.1 Alarm Logging

## 4.1.1 Active Alarm

The Android application only uses the read functionality when talking to the G4 device. 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:

*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.

## 4.1.2 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.

## Level Master Application

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. Register for total capacity for tank is AppSlot.122.x where AppSlot= LevelMaster Application AppSlot No and x= Tank Number. This register will return capacity in feet and inches.

Level Master application is also enhanced to add register for Unit of Temperature. The register number is X.121.Y where X is app slot and Y is tank number. Unit for volume is always Bbls.

These values for US devices is F and Bbls.

### 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.

#### Implementation Detail

Local Access implementation follows the MVC architecture with clear separation of the Model, Controller and View responsibilities.

Model is the data model classes which take care of data representation of various TotalFlow device interfaces.

Controller classes are mainly responsible for maintaining the overall flow and coordinate with the Model and View for correct visual representation on the web browser.

Views are representing the user interface layout and are maintained as HTML templates. These templates are updated with the help of jQuery to support dynamic data display.

## Templates

Templates are made to maximize the code reuse. These template files capture the UI elements which are used at multiple places. These templates are HTML (Ver 4.0 ) files containing the layout for various UI elements. Appropriate styling information for UI elements is maintained in the CSS files. This way, based on the screen requirement, one can choose particular Style information and then apply to the template which is appropriate to that screen as per its look and feel.

### Template Guideline

1. Capture the basic layout – This way the same HTML 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 HTML page (using the HTTP GET request) and then put the html content in one part of another HTML page. This part can be a Tab, Accordion or a simple div or span.
2. Define the basic UI components and serve as placeholder for the runtime data.
3. Ideally no style information should be present – Styling information is applied from jQuery, when the XML file is read and the page is updated. This way we have the maximum flexibility and the same template can then be used to show the same data in multiple look and feel formats as governed by the applied style (in Javascript file).
4. Styles information will be applied as classes. All such classes are maintained in CSS files.
5. 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 OperatorLayout.xml file is read.
6. Template files are read at different point of time. The html content of the template is given as input to other template/pages

### CSS

Styling information for the user interface part is maintained as separate files. CSS ( Ver 3) 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.

### JS based implementation

User Event and other dynamic handling is handled with the help of JavaScript. For basic validation, alerts, event handling, JS is used along with jQuery to handle and provide browser end page processing and support.

### jQuery

jQuery is used to support various features like Accordions, dynamic plotting of graph, periodic refresh support(using AJAX). Also, jQuery is used to do dynamic update of UI elements on screen and to move across the screens.

## Web Page layout

UI Implementation is done as per the screens in the OperatorUI.exe. Look and feel, UI components, layout and placing of UI controls, are done as per OperatorUI.exe. 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 64.

* 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, Tahoma Font, size 8 is used as the ‘font’ across the screens to display various text. . Focus is to use the ‘Tahoma’ font-style even in the third-party plugins. Such implementation depends upon as to how exactly font-style has been used in the plugins. In case, this change looks to be an overhead then the default font-styleof the plugin is used. 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 dynamic page (Example – Error in getting the data from device register using a HTTP request),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.

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 displayed with ‘??’ as the value for that particular device register on the web page. This behavior is to make the Local access web pages behavior consistent with other applications like PCCU which displays the device register value as ‘??’ when there is no value currently in a register.

Disclaimer

A disclaimer message is shown in the bottom of the login page that will capture following details:-

* + Resolution supported by the web pages
  + Name and version of the supported browsers along with recommended screen sizes

This disclaimer informs the user about the web pages compatibility and the right pre-requisite to access local access web pages. This helps to make sure that user accesses the web pages with correct browser environment for the better viewing experience and tested functionality support.

Resolution related disclaimer data is shown only if the resolution of the device in use is not matching the recommended size.

Help

In the current phase of the implementation, there is no support for any kind of Help pages. Based on the later requirements, the help page can be planned as any external link or internal web pages based help.

### Top Menu

Local Access Web pages have the menu which will provide options to move across various pages and perform various actions. This menu also contains information for ‘LogOff’ option (Underlined and non-bold font) and Current User Identifier (Welcome <User Name>) [ Font Size: 11px , Bold]. This menu is present across all pages (will conform to the menu options as displayed in OperatorUI.exe) and will conform to following specification:-

Menu Heading

* Font Style:- Tahoma
* Font Size: 18px

Menu Options

* Font Style: Tahoma, Underline
* Font Size: 14px
* Background Color: RGB (227, 239,255) OR #E3EFFF

### Bread Crumb

Bread Crumb is present across all pages. This section on top of page (below the menu) enables the user to know the user’s current location in the overall hierarchy of the Local Access Web pages. This breadcrumb section contains the links (left side of breadcrumb) to the page hierarchy and will enable the user to move up the hierarchy. So, this way user can easily move across the ‘Home’ Facility’ and Station level pages. Along with these page links, breadcrumb also contains the following information/option in the right side section of the breadcrumb area:-

* Last Updated Time ( displays the timestamp for the last page update )( Font is bold)
* Monitor ( Option to enable periodical update of the page )(Font is bold)
* Re-Read data ( Option to refresh the page with the current latest value )

Data in the breadcrumb section will conform to following specification:-

* Font Style: Tahoma, Underline (except Last Updated Time, Monitor)
* Font Size: 11px
* Font Color: RGB (21,66,169)

### Left Navigation

User can choose to browse across various screen sections using the Tab. This tab section is present on the left hand side 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. Mouse Hover (User mouse is currently over the tab)
3. Un-Selected (This tab is not the active tab)

This navigation section conforms to following specification:-

1. Un-Selected

* Font Style: Tahoma
* Font Size: 13px,
* Width : 90px
* Height: 30px
* Font Color: White
* Background Image (for unselected scenario)

1. Selected

* Font Style: Tahoma
* Font Size: 13px,
* Width : 90px
* Height: 30px
* Font Color: RGB(21,66,139)
* Background Image (for selected scenario)

1. Mouse Hover

* Font Style: Tahoma
* Font Size: 13px,
* Width : 90px
* Height: 30px
* Font Color: White

Background Image (for mouse hover scenario)

### Accordion

Local Access web page contains lot of data that is organized in accordions. The Accordion is a web 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 Title Date

* Font Style: Tahoma
* Font Size: 14 px,bold
* Height: 20px
* Font Color: Black
* Background Color: RGB (228,229,233)

1. Accordion Section Content

* Font Style: Tahoma
* Font Size: 11px
* Font Color: Black
* Background Color: RGB(235,236,239)

### Footer Section

In various Local Access web pages, bottom part of the page contains ‘Send’ and ‘Help’ button. Using the ‘Send’ button, user can send new/updated values of various fields to the Totalflow device. Using the ‘Help’ button, user can click on the help web pages and browse through them for the relevant section.

‘Send’ and ‘Help’ button(images to simulate buttons) conforms to following specification:-

* Height: 24px
* Width: 75px

Approach for showing the ‘Send’ button or not will depend if there is any editable field on the screen or not. In case there is no editable field, ‘Send’ button will be not shown. In case, there is any editable field, then screen will have the ‘Send’ button but in disabled state, which would get enable as soon as some value in the screen gets edited.

This enabling and disabling approach will be implemented by using an array which will track of the fields that got edited by user. So, every time any field is edited, it’s value along with register value is added to this array. A check is made to verify if the array already has this entry. This ensures that entries in this array for one particular field does not get duplicate and the latest change over rides the old entry. Send Button gets enabled only when there is at-least one entry in this array. If the user changes any field and tries to move out then he will be prompted to confirm if he will like to Send the changed values or not.

### Graphs

There will be graphs across various screens to capture and visualize the various parameters and monitor them over a period of time. With the help of 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. 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.

There will be two types of graph:-

* Static
* Dynamic

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

For implementing the graph in various screens, ‘Flot’ jQuery library is used.

### Tables

Tables are used to display the tabular data on screen. These tables are implemented using the ‘kiketable’ jQuery plugin. Tables adheres to following specification:-

* Header Cell Font (size in pixels) - Tahoma
* Header Cell Height:   20px
* Header Font: (size in pixels)  - 11px boldDraggable column to ensure variable width
* Minimum Column width (This minimum width will ensure that value with a minimum width in the table cell is displayed. This value will be 6 so that one can see value before and after the precision separator - ‘.’along with ‘…’ indicating that the value displayed is incomplete and one can the full value by viewing the field’s tooltip value).

## 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.