Assignment 2 CS456

Janson Webster (jswebste, 20277029)
Kuen Ching (tkching, 20273716)

Phase 1:
To detect neighbouring Bluetooth devices, we wrote a Search class and implemented it as a singleton. We made this design decision because it does not make sense for several Bluetooth adapters to exist in the application and attempt to scan from a single device. Inside our Search.java we have the listener whose responsibility is to populate a list and the logger file with the current scan's results when Bluetooth devices are discovered. This class provides all the functionality the UI thread needs to do any updates. Our UI thread required a handler to detect when the Bluetooth scan was completed to ensure that the Bluetooth device list is fully populated before the user is allowed to choose from the list. We decided to implement it this way as it does not make sense to update the list of devices as they are found in the scan. This will only cause confusion to the user if they are looking for a particular device which hasn't been found yet. By only posting the scan results once the scan has completed, this ensures the user knows for sure which Bluetooth devices are in range. The Bluetooth scan is also run in a dedicated background thread so that the UI will not become unresponsive while the scan is taking place.

Phase 2:
In this phase we created a class designed to query the server for which (if any) of the scanned Bluetooth devices are running the BLS application. When the user clicks on the "File List" button, the application will switch to a new activity which displays the MAC addresses found in Phase 1 which have registered on the BLS server. Once every Bluetooth MAC has been sent to the BLS server, the MAC addresses of those which have registered for the BLS service are then displayed in a scrollable list in the new activity. We decided to only display the MAC address of those devices which have registered for the BLS service as it makes it easier for the user to narrow down the possible devices they can connect to. If they want to know the MAC addresses of all Bluetooth devices found, this is already displayed on the main screen. We also decided to display the MAC addresses of the devices and not the IP addresses, as depending on what network the device is on, its associated IP address will be different. This means you can consistently determine which one is "Jimmy's" device. As MAC addresses are static, no matter which network the device is on, the user will always be able to determine which (if any) is "Jimmy's device". We created a new activity for this page so when the user presses the back button, the original activity (page) is left exactly as they left it. We chose to decouple the Bluetooth scan form the BLS server query so that the user can use the device to perform a Bluetooth scan even if they are not connected to the WiFi. Then once they connect to a WiFi network, they can then query the BLS server. The BLS lookup is run in a dedicated background thread to ensure that the UI will not become unresponsive while the lookup is taking place.

Phase 3:
In this phase, we created two separate socket classes which derive from a common abstract base socket class. One of these classes is for the server socket and for the client socket. Both of these classes are implemented in their own thread so that they can run in the background without freezing the UI. The server socket's class is responsible for specifically opening a port and listening to that port for clients requesting a connection. The server socket will only listen when

the user specifically tells the application it is ok to receive connections via the “Enable/Disable FT” button. When the user chooses to terminate the server connection, the server elegantly terminates any opened ports and existing sockets and then the thread is killed. When the server receives a connection from a client, it listens for the correct protocol to take place which is described below. For the UI part of this, the user must decide whether or not to enable the file list transfer. If it is off, the user must turn it on before other clients can view their file list. This is to not allow access when the user does wish to share it. On application start-up, the default setting for the transfer listener is off, forcing the user to explicitly turn it on for privacy reasons.

The client first sees if the device has the same protocol he does, and if it does it then continue asking for the file list. Similarly to the server socket, it outputs an error if any occur and logs all the action to the SD card. It can also propagate the errors to the user to let them know when an error occurs and possibly suggest a solution. The file list request occurs when the user click on one of the MAC addresses listed which are associated with the BLS. If the client receives a file list, the UI will change to a new activity displaying the files in a scrollable list. We chose to implement this as a new activity so that when the user presses the back button, they will go back to the BLS MAC list and can optionally chose to view the file list from another device. The client writes to the trace file regarding whether the file list request passed or failed and the file list if it succeeded. If it did not connect to the other device, the client will tell the user it couldn’t connect, and it should try again. This allows the user to still have access to all the MAC addresses it initially found without rescanning for nearby devices.

The protocol our application uses starts off with the server listening to a port. Then the client will send a socket to the server and say hello. The server recognizes the hello and confirms that it’s what it was expecting and then sends to the client an acknowledgement that it is connected. If the hello message is incorrect, the server kills the socket and tries to bind with another socket. Assuming the server sends an acknowledgement, the client will then need to say at any point to send the file list over. Once that happens, the server will need to run file list and then send it over to the client to read. If the server disconnects, the client will realize that it didn’t get the end of message so an error occurred and it should try again. Afterwards the server will send an end of file command and the client will say thanks and then close socket. Server’s socket will close as a result and then re-open and listen to the same port waiting for another device until closed.

Phase 4:
The file list generator starts at the root of the SD Card and iterates through all the files and folder in the root list. When it encounters a directory, the method will recursively call itself making this directory the new root. This process continues until all files are accounted for. The list of files is then sorted by full file path and a newline delimited string of the full file paths is returned. This method is used by the server socket to grab the file list from the SD Card when requested by a client. Since we can assume there is always a SD Card there are no error checks for this method.

Phase 5:
The logger enables the application to output what it’s doing to more details. Specifically to the use case of users and see how they are using the application. This writes to a file called “log.txt” located on the SD Card. It writes to the SD Card instead of the internal memory since there is more space on the SD Card than internal memory. The logger is designed to be a singleton as we

only want one open write connection to a file at any given time, otherwise there will be race conditions occurring in writing to the file, causing the log file to become corrupt.

Phase 6:
For the miscellaneous items of UI, we chose the last Bluetooth scan time to be represented as when the user pressed the start scan. This way there is an immediate update and the user knows that the scan is running. The status textbox will always reflect the last action which has taken place

On our main screen, the IP address will change dynamically as the device enters and exits areas of WiFi connectivity. It will also display the message "Not Connected" if the device is currently not connected to any WiFi network. We implemented it this way so the user will always know if the application has access to the WiFi, and thus can request a file list.

On the BLAS MAC address UI page (entered by clicking on "File List"), the user will get to see a status bar, and the MAC addresses that are recognized by the Bluetooth server as explained in Phase 2.   This page can only be accessed if there was at least one device discovered during the Bluetooth scan. Otherwise the button to get here will be disabled.

After selecting a MAC address that is valid the user will be presented a page that contains that device's file list. If not it means an error happened and the user will be told by a dialog box and will get to choose another MAC address. After the user sees the file list they will go back a page, and decide to see other devices, or they can go to the home screen and do a brand new scan of devices and repeat the process.