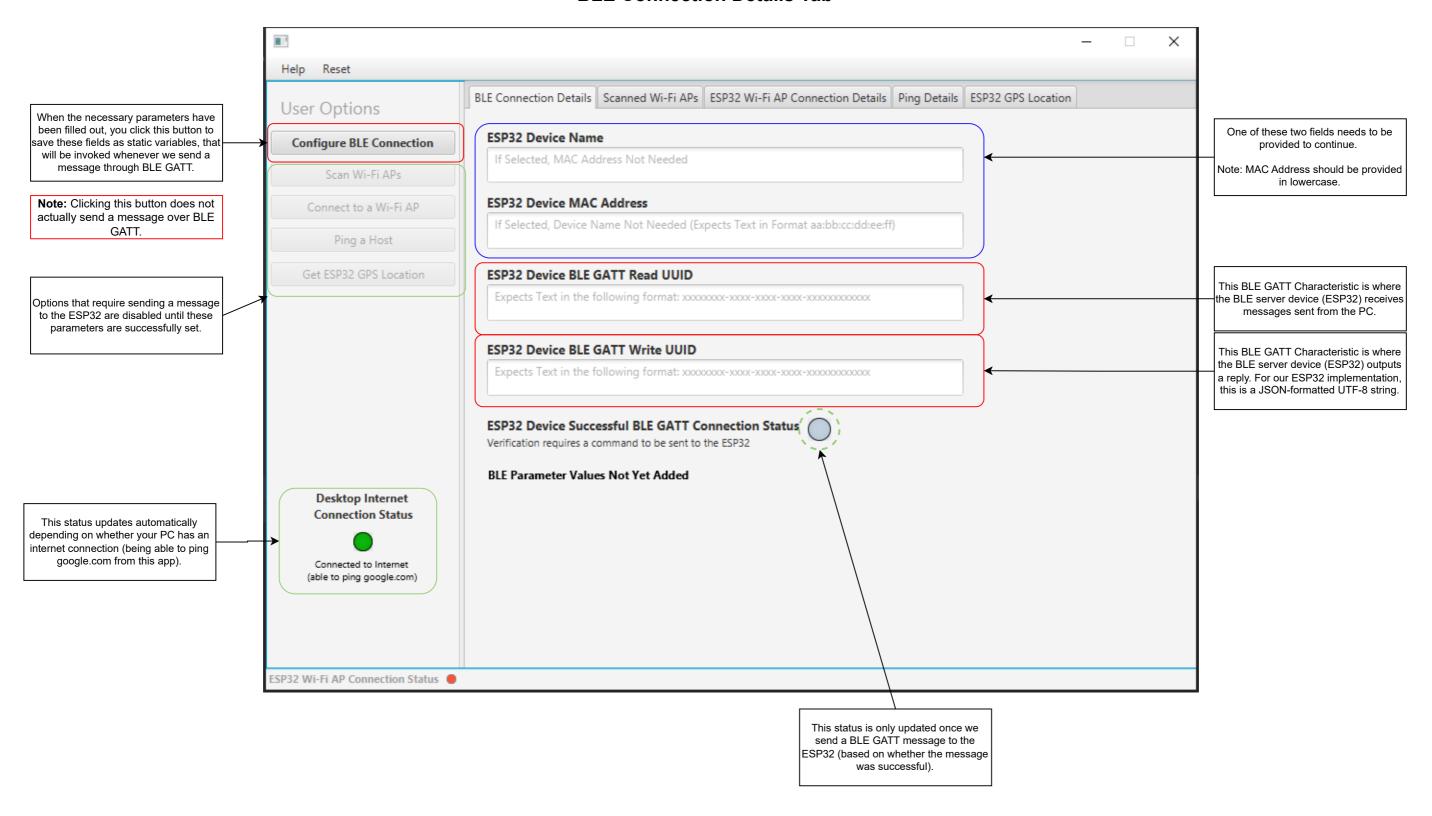
Project Applications Coordination

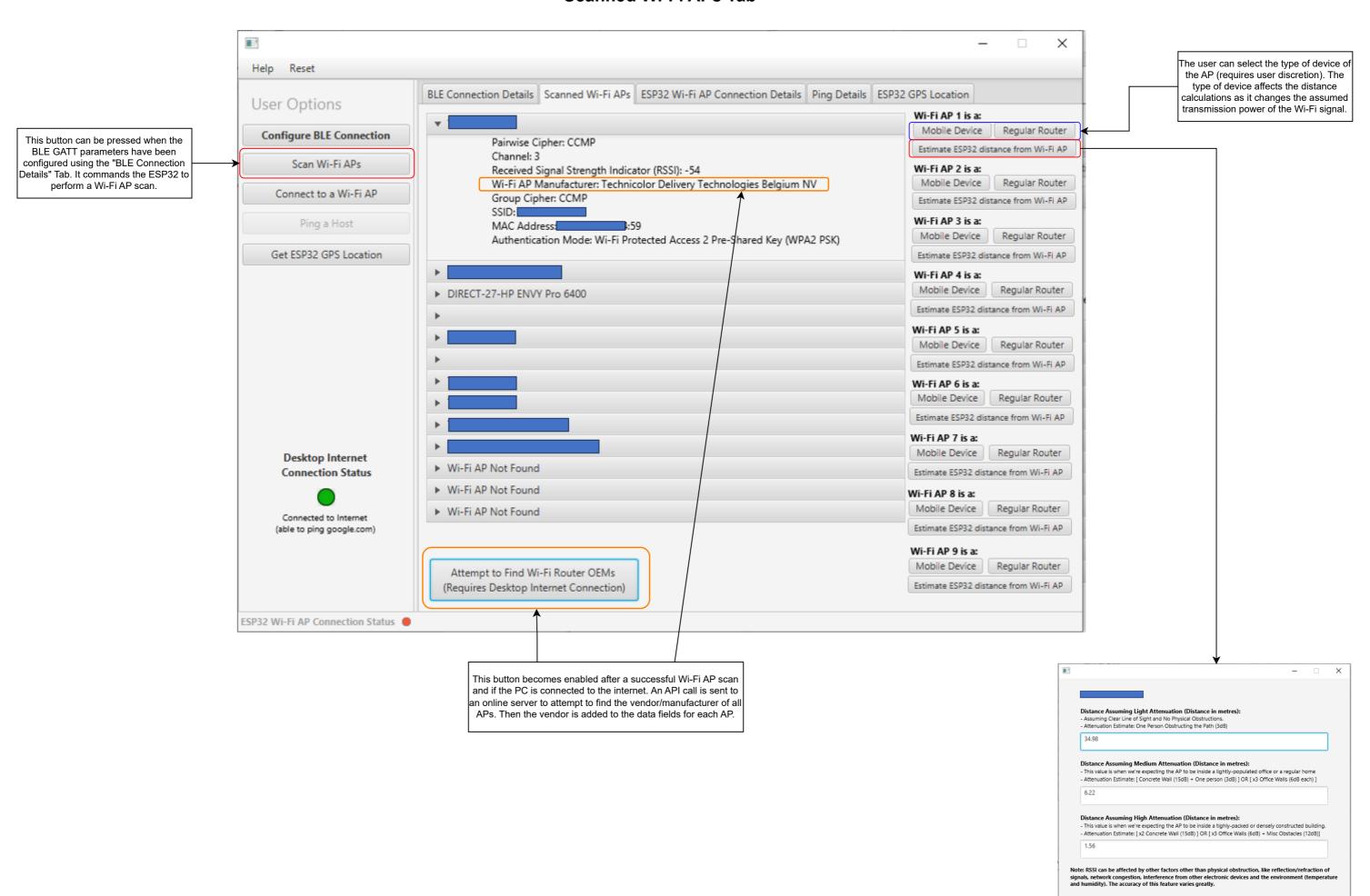
For this appliation to work, several Software projects (links given on Project README.md) need to be run and configured correctly.

Windows PC ESP32 Wi-Fi ESP32 Data Acquisition **BLE GATT Client for** Wi-Fi Data Acquisition Server Companion App Windows (Binary loaded in ESP32) (Executable App) (Executable App) Communicates with a BLE-Commands sent from this app are sent using RESTful JSON-Hosted in: (localhost:5900) enabled client by using certain pre-defined GATT formatted API requests. . Characteristic UUIDs. The user chooses to run this app as an API server. Provides several basic Wi-Fi The RESTful API server allows functions like scanning APs and connecting to APs. Also provides GPS coordinates (as other applications to configure BLE GATT messages to send to a selected BLE-enabled the ESP32 is meant to be device (and its specific GATT connected to a NEO-6M GPS Characteristics). module and wired up as stated - HTTP Requests → BLE in the corresponding project This application uses the page). existing Windows Bluetooth stack to send BLE messages.

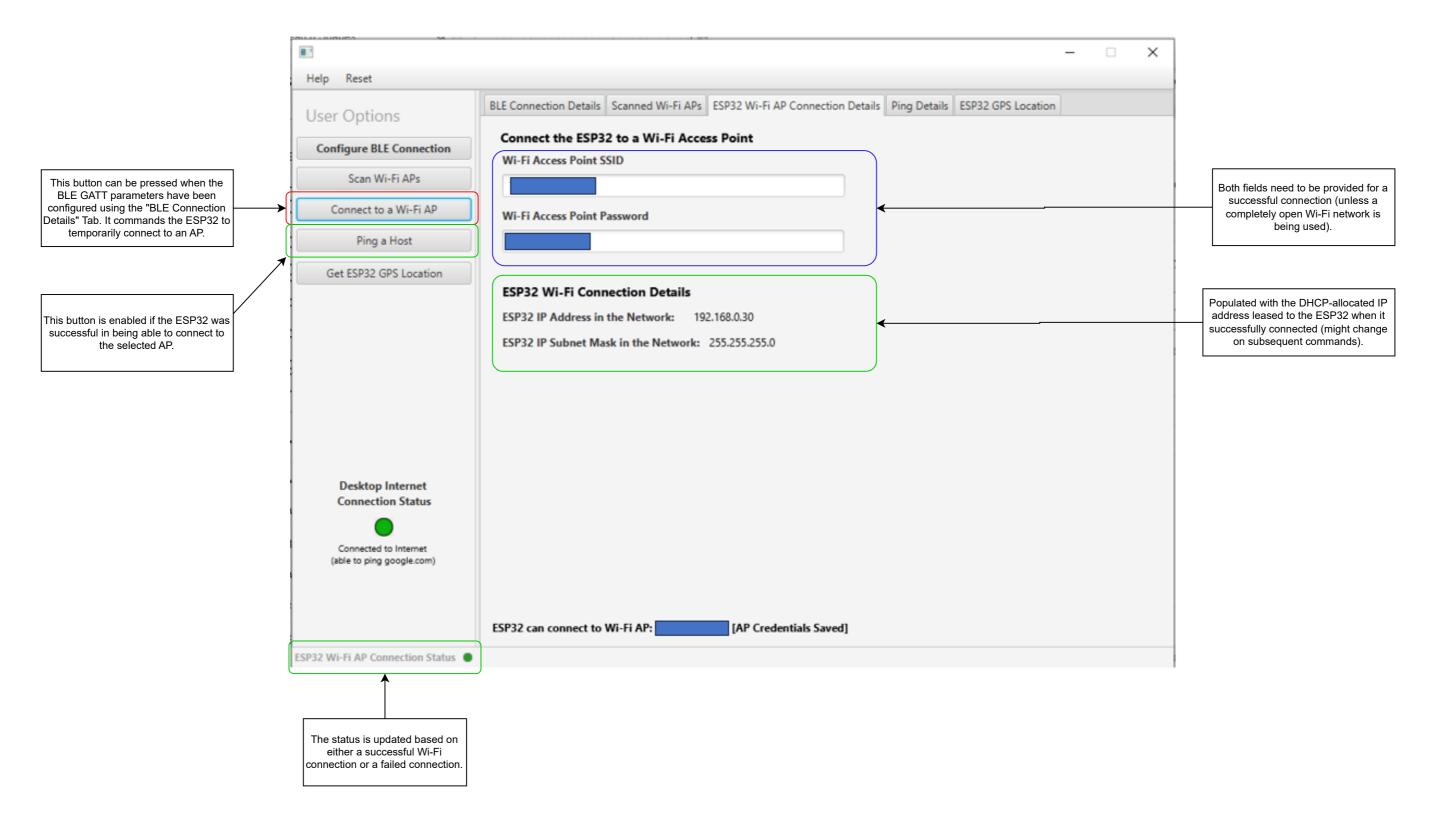
BLE Connection Details Tab



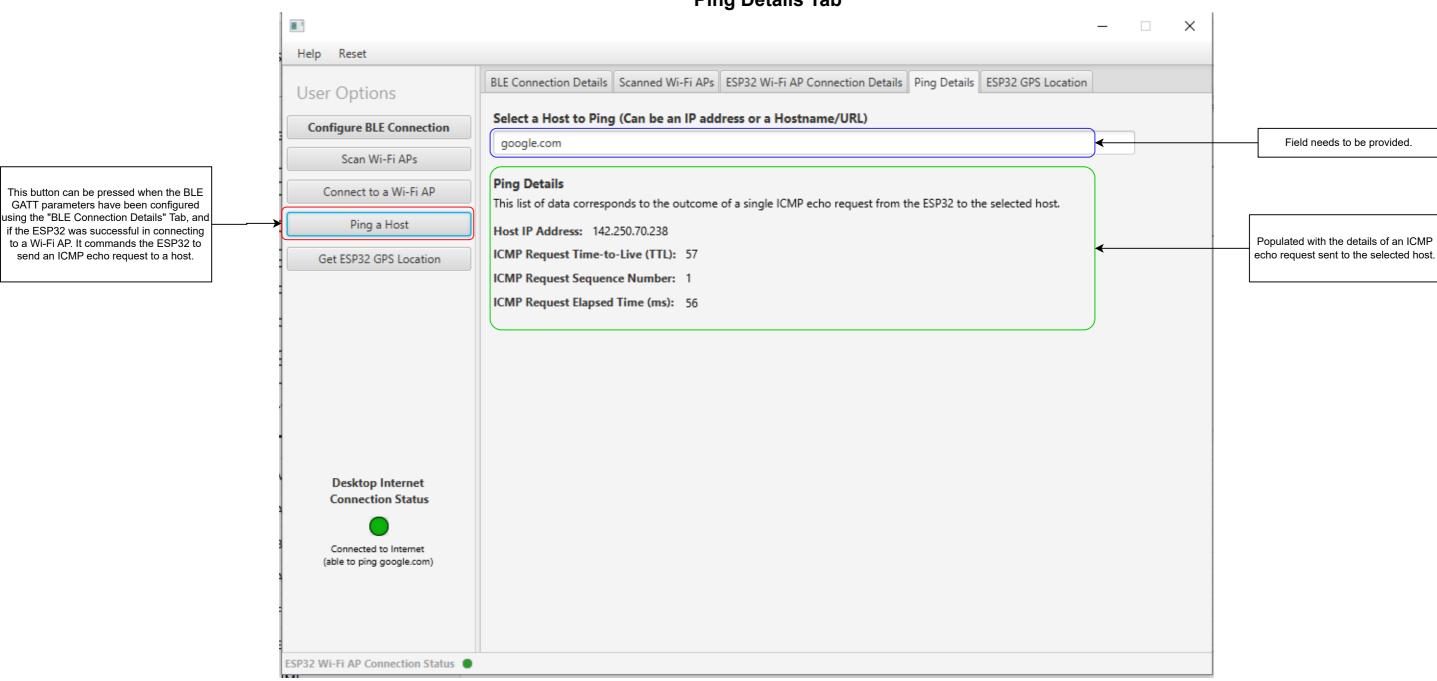
Scanned Wi-Fi APs Tab



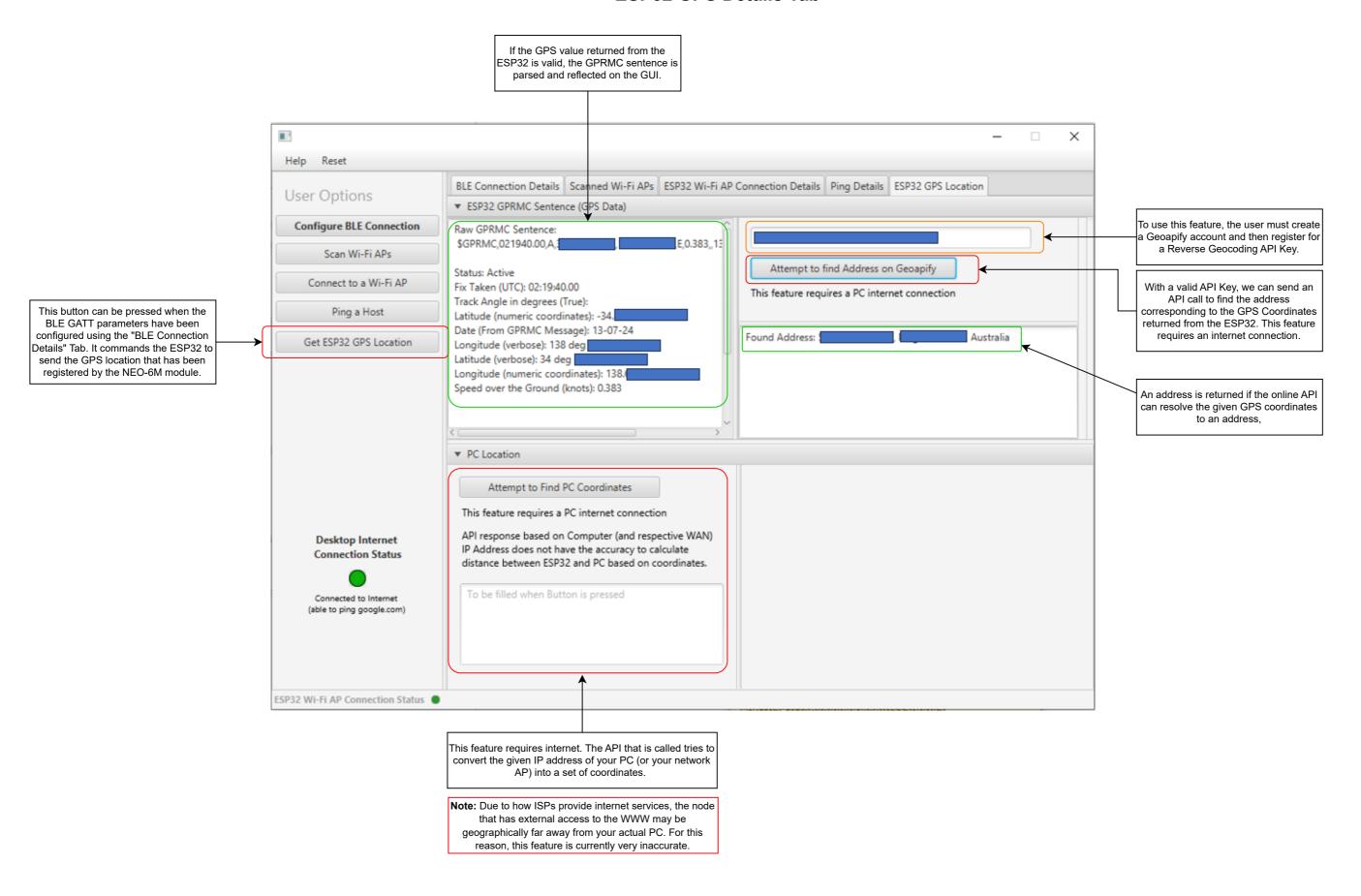
ESP32 Wi-Fi AP Connection Details Tab

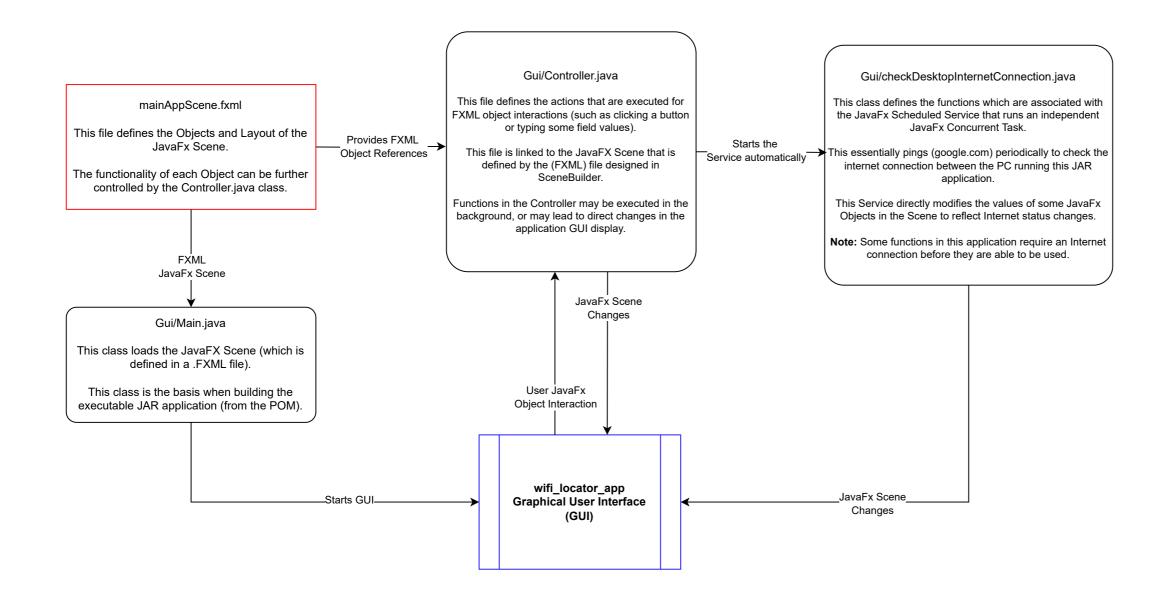


Ping Details Tab



ESP32 GPS Details Tab





Gui/Controller.java

click_configure_ble_button: This function does not actually send a message to the "BLE GATT Client for Windows.exe" application. It stores the BLE GATT parameters which are required by other functions when connecting to the "BLE_GATT_Client_for_Windows.exe" app.

- Ensure that functions (buttons) that are dependent on a successful BLE Connection are disabled.
- Stores the BLE parameters entered by the user into Static variables which can be accessed from any other class via the get/set functions.
- · Check if the minimum BLE parameters have been entered by the user, reenables other functions if the BLE parameters are determined to be OK (this is parameters).

a very limited check, since the API server has its own criteria for valid • Switch to the correct BLE GATT Configuration Tab on the GUI. User Clicks the "configure_ble_button" wifi_locator_app Graphical User Interface (GUI) The user is required to fill out the following text fields before being able to click the "Configure BLE" button (minimum requirements): BLE Device Name/MAC BLE GATT Read UUID BLE GATT Write UUID

outgoingApiCaller/bleConnection.java (Instance Name = applicationBleConnection)

A single instance of this class is created for the entire runtime of this App. This is because there are many Static variables associated with this class, and it is easier to keep them grouped.

multiple "set" functions for BLE parameters:

· Used to update the status of Static variables associated with the instance of this class with the values the user has entered in the JavaFx GUI.

check_mandatory_ble_parameters:

-Function Calls-

Function Call/Status

JavaFx Scene

Changes (Re-Enables Buttons that require valid BLE parameters first)

• Based on the static BLE connection parameters associated with the instance of this class, we determine if the necessary values are not empty (this is the only criteria for valid BLE parameters because the API server has its own criteria for what is acceptable formatting).

Gui/Controller.java

click_scan_wifi_ap:

- First we execute the Wi-Fi Scan in the background by calling function (execute_wifi_scan).
- Once we extract the string reply from our WiFi AP scan, we know the output is being provided in a custom JSON format (otherwise we interpret it as an error). We convert the reply to JSON for easy parsing.
- We store a copy of the array of all the JSON strings deected by the ESP32 (each String in the array represents one Wi-Fi AP). These values are <u>needed for finding the OEM for</u> <u>each AP</u>.

We iterate over every individual Wi-Fi AP (each defined by a single JSON object).

- We format the JSON key-value pairs into a more user-friendly format to display in GUI.
- We update the GUI JavaFx Objects to show the Wi-Fi Scan Data.

User Clicks the JavaFx Scene
"wifi_scan_option_button" Changes

wifi_locator_app

wifi_locator_app Graphical User Interface (GUI)

The user only needs to click the button (assuming the necessary BLE parameters have been successfully entered beforehand).

apiMsg/apiMsgConstructors.java

create_wifi_scan_msg:

 Creates a pre-defined JSON-formatted "wifi_scan" message that will trigger the ESP32 running the "wifi-data-acquisition-server" software to execute a Wi-Fi Access Point scan.

outgoingApiCaller/wifiScan.java (Instance Name = applicationWifiScan)

A single instance of this class is created for the entire runtime of this App. This is because there are many Static variables associated with this class, and it is easier to keep them grouped (associated with the same instance).

execute_wifi_scan:

-funct call-

- First we check if the necessary BLE parameters have been entered prior to creating the "wifi_scan" message.
- We create the pre-defined Wi-Fi Scan message by calling the (create wifi scan msg) function.
- At this point, we have the message to send over BLE, as well as the necessary parameters to communicate with the ESP32 over BLE. We call function "send_new_ble_msg_to_esp32" of the bleConnection Class to send our defined message over BLE GATT.
- We check if the reply from the ESP32 indicates that the message was successfully sent. A successful reply from the ESP32 is always JSON-formatted, if the reply was not a JSON string, then it must be an error message from the BLE Client App.

get_reply_str:

func call/

reply String

Returns the String containing multiple JSON "Objects".
 Each Object represents a Wi-Fi AP.

outgoingApiCaller/bleConnection.java (Instance Name = applicationBleConnection)

A single instance of this class is created for the entire runtime of this App. This is because there are many Static variables associated with this class, and it is easier to keep them grouped (associated with the same instance).

send_new_ble_msg_to_esp32:

function call/

String reply

func call/

String reply

Using the BLE GATT parameters that the user entered using the Java App GUI, we have the necessary info we need to send data to the "BLE_GATT_Client_for_Windows.exe" app, which communicates with the ESP32 running the "wifi-data-acquisition-server" software.

function calls/String replies

outgoingApiCaller/outgoingApiCaller.java

multiple "execute api call" function calls:

- Each "execute_api_call" defines a URL-encoded String using the (return_url_encoded_str) function.
 This is necessary in case there are any special characters in the user-defined string, URL encoding automatically converts those special characters to a compatible equivalent.
- The URL refers to the base "BLE_GATT_Client_for_Windows.exe" API server and any additional arguments/topics.
- Eventually, we call function (generic_http_request) with the URL we have defined to communicate with the BLE GATT API server.



This button is only enabled if the user:

- Has made a successful Wi-Fi AP scan with the ESP32
- · Has an internet connection.

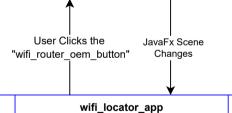
Gui/Controller.java

click_wifi_ap_oem_button:

· Each Wi-Fi AP's details are saved in a JSON String when the user executes a Wi-Fi scan. Each of these JSON strings are saved in an Array (latest set of scanned wifi aps).

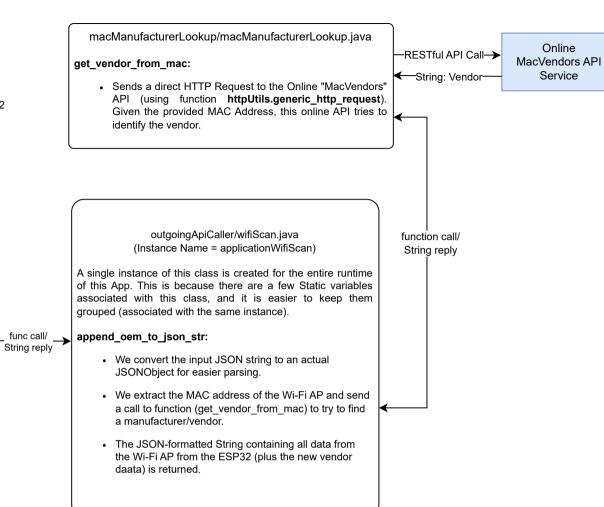
The following steps are repeated for each scanned Wi-Fi AP:

- We format the JSON key-value pairs into a more user-friendly format to display in GUI.
- We take this user-friendly version of the AP JSON string, and add the OEM as a new data point.
- · We update the AP TitledPane with the updated data from the JSON String (modified to no longer be shown as JSON).



Graphical User Interface (GUI)

The user only needs to click the button (assuming the necessary BLE parameters have been successfully entered beforehand).



apiMsg/apiMsgConstructors.java

 Creates a pre-defined JSON-formatted "wifi_conn" message that will trigger the ESP32 running the "wifidata-acquisition-server" software to temporarily try to connect to a Wi-Fi AP.

String reply

outgoingApiCaller/bleConnection.java
(Instance Name = applicationBleConnection)

func call/

A single instance of this class is created for the entire runtime of this App. This is because there are many Static variables associated with this class, and it is easier to keep them grouped (associated with the same instance).

send new ble msg to esp32:

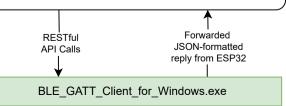
 Using the BLE GATT parameters that the user entered using the Java App GUI, we have the necessary info we need to send data to the "BLE_GATT_Client_for_Windows.exe" app, which communicates with the ESP32 running the "wifi-data-acquisition-server" software.

function calls/String replies

outgoingApiCaller/outgoingApiCaller.java

multiple "execute api call" function calls:

- Each "execute_api_call" defines a URL-encoded String using the (return_url_encoded_str) function. This is necessary in case there are any special characters in the user-defined string, URL encoding automatically converts those special characters to a compatible equivalent.
- The URL refers to the base "BLE_GATT_Client_for_Windows.exe" API server and any additional arguments/topics.
- Eventually, we call function (generic_http_request) with the URL we have defined to communicate with the BLE GATT API server.



Gui/Controller.java

click connect to ap button:

- We do a few preliminary checks to ensure that the SSID and password have been provided.
- We execute the function (execute_wifi_connection).
- We execute the function (get_reply_str) to display the relevant specific status updates in the "ESP32 Wi-Fi Connection Details" Tab of the Main application JavaFx Scene.

User Clicks the JavaFx Scene "wifi_conn_execute_button" Changes

wifi_locator_app Graphical User Interface (GUI)

The user is required to fill out the following text fields to configure the connection:

- · Wi-Fi Access Point SSID
- · Wi-Fi Access Point Password

outgoingApiCaller/esp32WifiConnection.java

This Class only contains static methods, which can be executed independently from a specific object instance.

execute_wifi_connection:

func call/

int status

__JSON-format

String reply

- First we check if the necessary BLE parameters have been entered prior to creating the "wifi conn" message.
- We create the pre-defined Wi-Fi Scan message by calling the (create_wifi_connect_msg) function.
- At this point, we have the message to send over BLE, as well as the necessary parameters to communicate with the ESP32 over BLE. We call function "send_new_ble_msg_to_esp32" of the bleConnection Class to send our defined message over BLE GATT.
- We check if the reply from the ESP32 indicates that the message was successfully sent. A successful reply from the ESP32 is always JSON-formatted, if the reply was not a JSON string, then it must be an error message from the BLE Client App.

get reply str:

 This function retrieves the latest reply from the ESP32 regarding the state of the connection attempt.

Gui/Controller.java click_esp_gps_button: func call/ We execute the function String reply (request gps gprmc data value). func call/ JSON Object We execute function (create gprmc json obj). · We use the JSON object representing the GPRMC data point and update the relevant GUI elements. User Clicks the JavaFx Scene "esp gps location button" Changes wifi locator app **Graphical User Interface (GUI)** The user is only required to press the button (assuming the necessary BLE parameters have been successfully entered beforehand).

apiMsg/apiMsgConstructors.java

create_gps_msg:

 Creates a pre-defined JSON-formatted "gps_location" message that will trigger the ESP32 running the "wifidata-acquisition-server" software to send the current GPS data stored in the buffer.

outgoingApiCaller/gpsCaller.java (Instance Name = esp32GpsCaller)

A single instance of this class is created for the entire runtime of this App. This is because there are a few Static variables associated with this class, and it is easier to keep them grouped (associated with the same instance).

request_gps_gprmc_data_value:

- We call function (execute_gps_location_request). This function composes a JSON-formatted message to forward to the ESP32.
 - We call function (create_gps_msg) to compose the JSON-formatted message.
 - At this point, we have the message to send over BLE, as well as the necessary parameters to communicate with the ESP32 over BLE. We call function (send_new_ble_msg_to_esp32) of the bleConnection Class to send our defined message over BLE GATT.
- The JSON-formatted String from the ESP32 is stored in static variable "reply str".
- The raw JSON string contains multiple GPS datapoints in varying GPS data formats, for our purposes, we only extract a singular GPRMC datapoint using function (extract gprmc data from msg).
- · We return the entire GPRMC data point as a String.

create_gprmc_json_obj:

 The GPRMC data point contains several different values which provide useful GPS info. We parse the GPRMC data into a JSON object providing convenient key-value pairs.

multiple func calls

gpsProcessing/gpsParser.java

multiple parsing functions:

• Each key-value pair in the final JSON object must be carefully extracted from the raw GPRMC string.

outgoingApiCaller/bleConnection.java
(Instance Name = applicationBleConnection)

A single instance of this class is created for the entire runtime of this App. This is because there are many Static variables associated with this class, and it is easier to keep them grouped (associated with the same instance).

send new ble msg to esp32:

func call/

String reply

 Using the BLE GATT parameters that the user entered using the Java App GUI, we have the necessary info we need to send data to the "BLE_GATT_Client_for_Windows.exe" app, which communicates with the ESP32 running the "wifi-data-acquisition-server" software.

function calls/String replies

outgoingApiCaller/outgoingApiCaller.java

multiple "execute api call" function calls:

- Each "execute_api_call" defines a URL-encoded String using the (return_url_encoded_str) function.
 This is necessary in case there are any special characters in the user-defined string, URL encoding automatically converts those special characters to a compatible equivalent.
- The URL refers to the base "BLE_GATT_Client_for_Windows.exe" API server and any additional arguments/topics.
- Eventually, we call function (generic_http_request) with the URL we have defined to communicate with the BLE GATT API server.



BLE GATT Client for Windows.exe