EzeFind
System Documentation

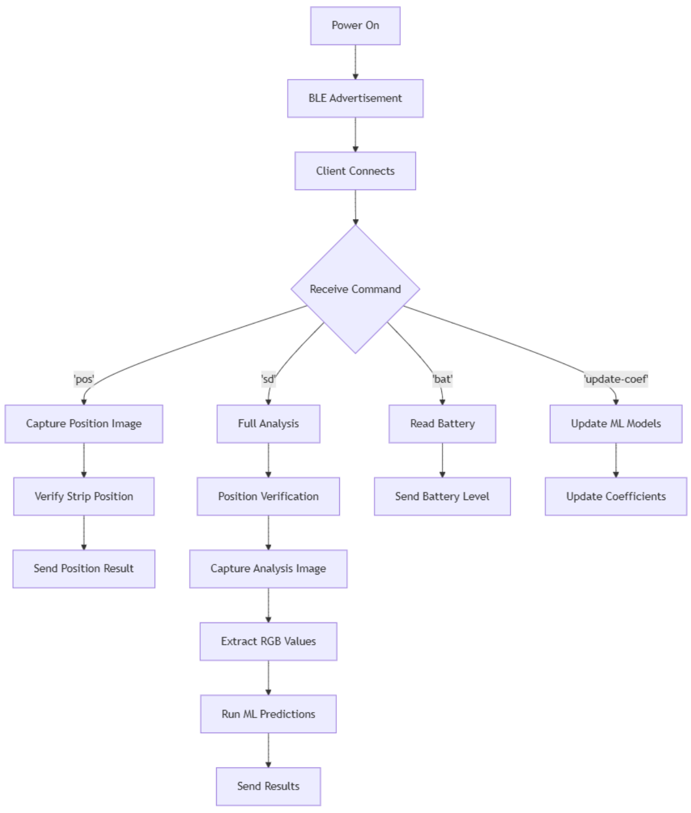
---

1. System Workflow

The EzeFind device is a portable urine-strip analysis system built on a Raspberry Pi with an integrated camera and Bluetooth Low Energy (BLE). On power-up, the device initializes its hardware modules (camera, RGB processing, position verification) and registers with the BlueZ BLE stack to advertise itself as a peripheral. A custom BLE service is created with several characteristics for command-and-control, data transmission, and image transfer. The device waits in an idle state, broadcasting its BLE name, until a client (e.g. a smartphone app) connects. Once connected, the UnitCharacteristic (UUID) listens for write commands from the client. These commands include:

- ***pos*** – Verify the position of the urine test strip in the camera frame.
- ***sd*** – Perform the full strip scan: verify position and capture RGB image data.
- ***bat*** – Read and report the battery level.
- ***update-coef*** – Begin a multi-part update of calibration coefficients (followed by JSON data chunks).

Workflow Overview

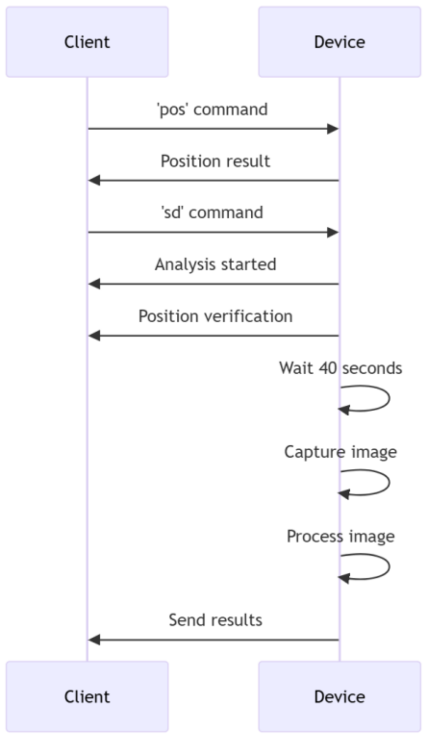


In summary, the workflow is:
Startup: Initialize camera, RGB module, position-verifier, and BLE agent (pairing disabled).
BLE Advertising: Broadcast a device name, then await connection.
Client Connection: Upon BLE client connect, the client send commands (pos, sd, bat, etc.) via the UUID.

- **Position Check (pos):** Camera captures a test image, and pos.ver_pos() analyzes it to return "true" or "false". The result is sent back via notification.
- **Strip Scan (sd):** Wait ~20 s, capture image for first position check. If not aligned, retry after another delay. Once confirmed, capture full strip image, use RGBModule to compute analyte colors, and prepare the full-color JPEG for transfer.
- **Battery Check (bat):** Read battery voltage and send as a notification.
- **Data Transmission:** The client subscribes to notifications on the PositionCharacteristic (for text data) or ImageCharacteristic (for raw image bytes). The device sends the requested data, splitting large images into BLE packets with headers and EOF markers.

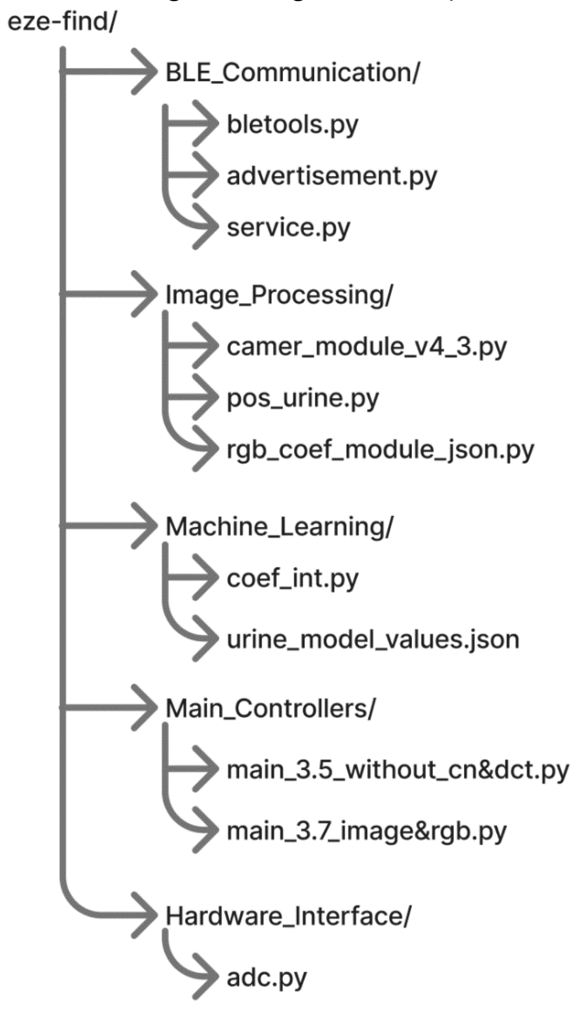
- **Coefficient Update:** If update-coef is sent, subsequent writes supply JSON chunks of calibration data. Once complete ("etx" command), the device updates its urine-analysis coefficients.

Analysis Workflow



2. Code Structure

The EzeFind codebase is organized into hardware interface modules, BLE service modules, and the main control logic. The high-level components are:



```
eze-find/
├── BLE_Communication/
│   ├── bletools.py
│   ├── advertisement.py
│   └── service.py
├── Image_Processing/
│   ├── camer_module_v4_3.py
│   ├── pos_urine.py
│   └── rgb_coef_module_json.py
├── Machine_Learning/
│   ├── coef_int.py
│   └── urine_model_values.json
├── Main_Controllers/
│   ├── main_3.5_without_cn&dct.py
│   └── main_3.7_image&rgb.py
└── Hardware_Interface/
    └── adc.py
```

BLE_Communication:

- bletools.py – Utility functions for BLE; provides get_bus() (to get the DBus SystemBus) and find_adapter() (to locate a BLE adapter).
- advertisement.py – Defines an Advertisement class to register BLE advertisements.
- service.py – Implements generic GATT server classes (Application, Service, Characteristic, Descriptor) using D-Bus. These wrap the BlueZ GATT APIs to create custom BLE services and characteristics (e.g. DeviceService, ImageCharacteristic).

Image_Processing:

- camer_module_v4_3.py – Provides CameraModule which wraps PiCamera2 to capture images. It handles auto-exposure adjustment and optional color correction, and can return either raw image arrays or JPEG byte data.
- pos_urine.py – Contains the pos class that analyzes a captured image to verify the position of the urine strip. The method ver_pos(image) saves a timestamped image file and uses OpenCV processing (grayscale, blurring, thresholding, morphology) to detect strip edges and return 'true' or 'false'.
- rgb_coef_module_json.py – Implements RGBModule, which loads calibration coefficients from urine_model_values.json and processes strip images to extract RGB values and predict analyte concentrations. The core function get_rgb_values(image) uses image segmentation and block averaging to measure each test pad's color, then applies linear models to compute PRED values for Leukocytes, pH, SG, etc. The results are returned in a dictionary.

Machine_Learning:

- coef_int.py – Contains commented-out Python versions of the calibration coefficients (for pH, SG, WBC, blood, etc.) that mirror the JSON data. This serves as a reference but is not actively used at runtime.

Main_Controllers:

- main_3.5_without_cn&dct.py – A variant of the main program without the image-transfer characteristic (ImageCharacteristic is omitted). It uses only Position and Unit characteristics for a simpler workflow.
- main_3.7_image&rgb.py – The primary program. It configures the BLE agent (disables pairing via NoPairAgent), initializes global managers (state manager, hardware modules), defines BLE advertisement and a custom service (DeviceService with Position, Unit, and Image characteristics), and runs the main event loop. All interaction logic (handling commands, capturing images, etc.) is implemented here.

Hardware_Interface:

- adc.py – Defines ADS1115Sensor for reading voltages via an ADS1115 ADC over I^{2}C. Currently used for reading battery voltage (though in main code the battery reading is stubbed).

3. Usage Instructions

- **Initial Setup:**

```
# Install dependencies
pip install picamera2 opencv-python numpy dbus-python adafruit-circuitpython-
ads1x15
```

4. File and Module Descriptions

Below is a detailed description of each file/module, its purpose, and key components:

~ ***advertisement.py***

- **Purpose:** Provides BLE advertising functionality for BlueZ.
- **Key Class:** Advertisement extends dbus.service.Object. It creates a BLE advertisement object with properties like type (peripheral), local name, service UUIDs, etc.
- **How It Works:** On instantiation, it sets the path and stores the advertising parameters. The method add_local_name(name) sets the advertised device name.
- **Integration:** In the main program, a subclass BLEAdvertisement calls add_local_name and then register() to start advertising with BlueZ. The register() method uses LEAdvertisingManager1. RegisterAdvertisement to make the device discoverable.

~service.py

- **Purpose:** Implements the GATT server framework using D-Bus for BlueZ.
- **Key Classes:**
  - ***Application:*** Manages the main loop and holds services. Implements GetManagedObjects to provide DBus object paths of all services, characteristics, and descriptors. Methods register() and run() start the GATT service via RegisterApplication.
  - ***Service:*** Represents a GATT service with a UUID and a list of characteristics. It provides D-Bus properties for the service (UUID, primary, characteristic paths).
  - ***Characteristic:*** Represents a GATT characteristic. It has a UUID, flags (e.g. ["notify", "write"]), and associated descriptors. It provides default ReadValue, WriteValue, StartNotify, and StopNotify methods. It also emits PropertiesChanged signals for notifications.
  - ***Descriptor:*** Represents a GATT descriptor under a characteristic (e.g. the standard User Description). Provides default implementations of Read/Write that must be overridden if needed.
- **Integration:**
  service.py is used by main_*.py to create an Application instance and define a DeviceService (subclass of Service) containing our custom characteristics (PositionCharacteristic, UnitCharacteristic, ImageCharacteristic). These classes inherit from Characteristic or Descriptor defined here, overriding WriteValue, StartNotify, etc. The Application object handles registration with the BlueZ GATT manager.

~bletools.py

- **Purpose:** BLE utility functions for finding and powering the Bluetooth adapter.
- **Key Methods:**
  - ***get_bus():*** Returns the D-Bus SystemBus object.
  - ***find_adapter(bus):*** Searches all managed objects for an object path that supports the LEAdvertisingManager1 interface, indicating a BLE adapter.
- **Integration:**
  Used by Advertisement and Application to get the bus and locate the BLE adapter path when registering services or advertisements.

~main_3.7_image&rgb.py

- **Purpose:** The main application program (version 3.7) including image capture and RGB analysis.
- **Key Sections:**
  - ***NoPairAgent:*** Defines a BLE agent that rejects any PIN or passkey requests to disable pairing. The function register_no_pair_agent() registers this agent with BlueZ.
  - ***StateManager:*** Manages persistent device state (last command, last device) in device_state.json. It loads state on init and saves updates on each command.
- **Module Initialization: Instantiates global modules:** pos_v = pos() (strip verifier), cam = CameraModule(), rgb = RGBModule().
  - ***BLEAdvertisement:*** Subclass of Advertisement that sets the local name.
  - ***ImageCharacteristic (UUID 4d6783c9-...):*** Supports ["notify","read"]. Main methods:
    prepare_data(): Captures an image with cam.capture_image. Stores the JPEG bytes in self.data_buffer and calculates self.total_packets.
    get_next_packet(): Returns the next 200-byte slice of the image buffer, incrementing self.current_packet.
    StartNotify(): When notifications are enabled, sends a header (total size, packet count) as one notification, then schedules send_next_packet().
    send_next_packet(): Called repeatedly to send each image packet as a notification. After all packets, sends an EOF marker [0xFF,0xD9,0xFF,0xEE,0xEE] and logs transfer stats. Then clears the buffer.
    Descriptor ImageDescriptor: A read-only descriptor with value "Image Data".
  - ***PositionCharacteristic (UUID c2d2dfec-...):*** Supports ["notify","read"]. Handles position verification, RGB extraction, and battery:
    prepare_position_data(): Captures an image for position verification, saves a debug JPEG, runs pos_v.ver_pos(img), and sets self.position_result.
    prepare_rgb_data(): Performs the full-strip scan: waits 19 s, captures an image, verifies position (ver_pos), notifies the result. If result is "true", waits another 40 s and captures again for RGB. If "false",

waits 20 s to retry, notifies new result, then captures. It then calls rgb.get_rgb_values(img) to get all analyte values and stores them in self.rgb_data. It also invokes ImageCharacteristic.prepare_data() to ready the image for sending (via self.service.get_characteristic(...)).
prepare_battery_data(): Reads battery level (currently hardcoded to 1.81 V), sets self.battery_level.
clear_data(): Resets all prepared data flags after sending.
ReadValue(): If data is not prepared, returns a message prompting which command to send (e.g. “Send 'pos' command first”). Otherwise returns a status string of which data is ready.
StartNotify(): Sends the prepared data: if self.rgb_data is set, sends each as UTF-8 string packets followed by EOF; else if self.position_result exists, sends it; else if self.battery_level exists, sends it. Clears data afterward.

- ***UnitCharacteristic (UUID c1e53625-...):*** Supports ["notify","write"]. Used for commands and coefficient updates.
  WriteValue(value, options): Decodes the UTF-8 command string from value. Handles a state where self.awaiting_coef is True (accumulate JSON data chunks until "etx", then calls pos_characteristic.update_coef() with the full data). Otherwise, for new commands, it updates state_manager, then calls pos_characteristic.prepare_battery_data(), prepare_rgb_data(), or prepare_position_data() for commands "bat", "sd", and "pos" respectively.
  ReadValue(): Returns a default "Ready" message for read requests.
  UnitDescriptor: A read-only descriptor with value "Command Control".

- **DeviceService:** A subclass of Service with a fixed UUID (90bbc477-...). In its constructor, it creates and adds the three characteristics: PositionCharacteristic, UnitCharacteristic, and ImageCharacteristic. It also provides a helper get_characteristic(uuid) to retrieve a characteristic by UUID.
- **Main Loop:** The main() function registers the no-pair agent, creates an Application, adds the DeviceService, registers it with BLE, and starts advertising. Then it calls app.run() to enter the GLib main loop.

~main_3.5_without_cn&dct.py

- **Purpose:** A simplified main program (version 3.5) without image transfer. It otherwise mirrors main_3.7, except it omits the ImageCharacteristic.
- **Differences:** It defines only PositionCharacteristic and UnitCharacteristic and does not include ImageCharacteristic in DeviceService.
- **Behavior:** Supports the same commands (pos, bat, sd, update-coef) to verify position, read battery, and compute RGB data, but instead of packaging a full JPEG image, it simply calculates RGB values and sends them as notification packets. This mode may be used on devices without image-transfer capability or as an earlier prototype.
- **All other logic (NoPairAgent, StateManager, advertisement) is identical to main_3.7.**

~camer_module_v4_3.py

- **Purpose:** Interface to the PiCamera2 for image capture with automatic exposure and optional color correction.
- **Key Class:** CameraModule.
- **Initialization:** Creates a Picamera2 instance and configures it for still images at 800×600 resolution. Exposure mode is set to ‘auto’ with automatic gain control enabled.
- **capture_image(apply_color_correction=True, byte_image=True, filename="bright_image.jpg"):**
- Starts the camera and captures a quick preview to measure average brightness.
- Adjusts exposure time and gain to reach a target brightness (around mid-scale).
- Applies new controls (exposure, gain, brightness, contrast, color gains) and waits 2 seconds for them to take effect.
- Captures the final image array with self.camera.capture_array().
- Optionally applies color correction: it converts the image to float, computes average R,G,B, scales each channel to balance colors, and normalizes brightness. Also adjusts HSV brightness to match target.
- Saves the corrected image to disk (filename or with _corrected suffix) and converts the final image to JPEG bytes if byte_image=True.

  - Returns either the raw image array or JPEG bytes. Exceptions are caught and logged; on error it returns None.
- **get_image_size(image_bytes):** Returns the byte length (used for headers).
- **Integration:** The main program uses cam.capture_image() to take pictures. For position checks it uses byte_image=False to get a NumPy array; for image transfer it uses byte_image=True to get JPEG bytes ready for sending.

~pos_urine.py

- **Purpose:** Contains logic to verify that a urine test strip is correctly positioned in the captured image.
- **Key Class:** pos.
- **ver_pos(self, image):** Takes a NumPy image array (image) of the scene. It:
  - Clears any previous debug images on disk.
  - Saves the input image with a timestamped filename for debugging.
  - Converts the image to grayscale, applies median blur, and sharpens it.
  - Thresholds the sharpened image to create a binary mask of dark regions (likely the strip edges).
  - Scans row 50 from left and right to find the first and last dark pixel positions, cropping the image to the strip region.
  - On the cropped region, applies a second threshold and morphology pass.
  - Iterates through rows (from top of cropped area) in steps to detect contiguous dark blocks indicative of test pads. It builds lst_no of row indices where a block of expected size is detected.
  - If multiple blocks are found with spacing and count matching the known number of pads (10 pads, correct spacing), and reaching near the bottom of the strip, it prints 'true'; otherwise 'false'.
  - Returns the string 'true' or 'false' accordingly. Errors default to 'false'.
- **Integration:** Called by PositionCharacteristic.prepare_position_data() and prepare_rgb_data() to ensure the strip is in view before extracting colors. The console logs and saved debug images help verify operation.

~rgb_coef_module_json.py

- **Purpose:** Extract RGB color readings from a urine test strip image and compute predicted analyte values using calibration data.
- **Key Class:** RGBModule. On init, it loads urine_model_values.json (a dictionary of coefficients and block coordinates for each analyte).
- **update_coef(self, coef, ...):** Parses a JSON string coef representing calibration data. It processes certain fields into lists or floats, then writes the resulting dict back to urine_model_values.json and reloads it into self.urine_model_values. Used when receiving new calibration over BLE.
- **predict functions:** Implements prediction formulas for each analyte (pH, SG, WBC, etc.) based on linear models. For single-coefficient models, predict(). For multi-category models like WBC or blood, functions like predict_wb() choose the max response and map it to a grade (e.g. 0,25,75,500).
- **detect_block(img_crop, f_p, s_p, w1, w2):** Extracts a sub-block of img_crop given row/col bounds and returns the average [R,G,B].
- **get_rgb_values(self, image):** Main method to process a full strip image:
  - Saves the input image as input_image.jpg for debugging.
  - Similar to pos, converts to grayscale and thresholds to find strip bounds and crop to the strip region.
  - On the cropped image, applies another round of thresholding to locate the band rows.
  - If a valid set of rows (lst_no) is found (meaning the strip likely has 10 regions), it slices the cropped image into two parts (since some reagents may be on the left part of strip and others on right).
  - Using coordinates from the loaded JSON (self.urine_model_values['color_block_params']), it calls detect_block on each colored pad region.
  - It then builds results_dict mapping indices 0–9 to each analyte, storing the average RGB and the predicted result (using the above predict functions and the coefficients).
  - Prints "Successfully extracted all RGB values" and returns this dictionary. If detection fails (no rows or invalid spacing), it returns an empty dict.
- **Integration:** Called by PositionCharacteristic.prepare_rgb_data() after capturing the strip image. The resulting results_dict contains the analyte readings that an external app could read. In the current BLE

logic, after computing self.rgb_data, the device prepares the JPEG image for transfer and sends it; it may also send self.rgb_data as notification payloads.

~adc.py

- **Purpose:** Interface for an ADS1115 analog-to-digital converter to read voltages (e.g. battery level).
- **Key Class:** ADS1115Sensor.
- **Functions:**
  - On init, sets up I²C and configures gain. Creates AnalogIn channels 0–3.
  - read_channel(channel): Returns the measured voltage on that channel.
  - read_all_channels(): Returns a dict of voltages on all four channels.
  - monitor_channel(channel): Continuously reads a channel at a given interval (prints or callback).
- **Integration:** Intended for battery monitoring. In PositionCharacteristic.prepare_battery_data(), one would call this sensor to get the actual battery voltage. Currently the code hardcodes self.battery_level = 1.81, so the ADC is not yet used. To fully implement battery reading, replace the hardcoded value with ADS1115Sensor().read_channel(0) or similar.

~coef_int.py

- **Purpose:** Provides reference calibration values (coefficients and intercepts) for urine analytes.
- **Contents:** Commented-out Python variables for coefficient arrays (e.g. coef_ph, coef_wbc, coef_bld, etc.) and intercepts (int_ph, int_wbc, ...). These correspond to the data expected in urine_model_values.json.
- **Integration:** This file is not imported by the main code; it seems to serve as a backup or documentation of calibration values. Actual coefficients are loaded from JSON by RGBModule.