---

1. Programming BLE on RB-BLE BE33 (with RuggedBoard A5D2X)

What is RB-BLE BE33?

RB-BLE BE33 is a BLE module designed for integration with embedded systems like the **RuggedBoard A5D2X**. It typically communicates using **UART (AT commands)** or **HCI interface**.

Programming Options:

- **UART Interface (AT Command Mode)**:
  - Send `AT+SCAN`, `AT+CONN`, `AT+GATTWR`, etc. via UART.
  - Use `minicom`, `screen`, or a C program to send commands from A5D2X.
- **HCI Interface (Host Controller Interface)**:
  - Treat the module as a BLE controller and use a BLE host stack (e.g., BlueZ on Linux).

Tools:

- UART communication using `termios` in C or Python.
- Optionally use `BlueZ` stack on Linux to control BLE using `hcitool`, `bluetoothctl`.

---

2. GATT Profile & Services

What is GATT?

Generic Attribute Profile (GATT) defines how **BLE devices communicate data** using a structured hierarchy:

- **Profile** → contains **Services**
- **Service** → groups related **Characteristics**
- **Characteristic** → contains a **Value** and optional **Descriptors**

Example:

Profile	Service	Characteristic
Heart Rate	Heart Rate Svc	Heart Rate Measurement (read/notify)
Battery	Battery Service	Battery Level (read)

Bluetooth Low Energy (BLE) Stack Explained

The BLE stack is typically split into **two main parts**:

Host Layer – Software side (e.g., Linux/Zephyr)
Controller Layer – Hardware + low-level firmware (e.g., RB-BLE BE33)

---

1. BLE Host Layer

This part **runs on your processor** (e.g., A5D2X), and handles **data logic**, **device management**, and **profile handling**.

Layer	Description	Example (Linux / Zephyr)
Application	Your program logic (sensor data, control logic)	Your C app, Zephyr app
GATT (Generic Attribute Profile)	Defines data exchange structure (e.g., heart rate, battery level)	`bt_gatt_*` in Zephyr
ATT (Attribute Protocol)	Transfers values (read/write/notify characteristics)	Used internally
L2CAP	Logical link layer; handles multiplexing and fragmentation	`l2cap.c`
SMP (Security Manager Protocol)	Handles pairing, encryption, bonding	Passkeys, Just Works

2. BLE Controller Layer

This part is usually inside a **BLE module or SoC** (like the **RB-BLE BE33**). It manages **low-level radio tasks**.

Layer	Description
Link Layer (LL)	Controls advertising, connections, packet handling
Physical Layer (PHY)	Actual RF transmission over 2.4 GHz

Communication Between Host ↔ Controller

They communicate via **HCI (Host Controller Interface)** using:

- **UART**
- **USB**
- **SPI**

For example:

- On **RuggedBoard A5D2X**, you communicate with RB-BLE BE33 over **UART HCI** or **AT Commands**.
- On **Linux**, HCI is handled by **BlueZ**.
- On **Zephyr**, host and controller might both run on the **same chip**.

Summary of BLE Stack Flow

You (App) →
GATT →
ATT →
L2CAP →
HCI →
Link Layer →
PHY →
Over the air Practical Mapping to Your Setup

Component	Role in BLE Stack
Your C App	Application
BlueZ or Zephyr BLE APIs	Host Layer (GATT, ATT, etc.)
RB-BLE BE33 module	Controller (Link Layer + PHY)
UART	HCI transport between host & controller