

how to make an android app to communicate hc05 connected with
stm32f407

To make an **Android app that communicates with an HC-05 Bluetooth module connected to an STM32F407**, you need to handle three layers:

1. STM32 firmware (UART ↔ HC-05)
2. HC-05 Bluetooth Classic configuration
3. Android app (Bluetooth Classic SPP)

I'll walk you through the **complete, practical flow** from hardware to app.

1 Hardware & Connection (STM32 ↔ HC-05)

HC-05 Pins

HC-05	STM32F407
VCC	5V (or 3.3V if supported)
GND	GND
TXD	STM32 RX (e.g., PA10 – USART1_RX)
RXD	STM32 TX (e.g., PA9 – USART1_TX)
EN / KEY	Optional (for AT mode)

⚠️ Important

- HC-05 RX is 3.3V tolerant, but use a **voltage divider** if powering from 5V.
 - Baud rate default: **9600** (data mode)
-

2 STM32F407 Firmware (UART)

Use **HAL** or **LL** drivers.

UART Initialization (HAL Example)

```
C

UART_HandleTypeDef huart1;

void MX_USART1_UART_Init(void)
{
    huart1.Instance = USART1;
    huart1.Init.BaudRate = 9600;
    huart1.Init.WordLength = UART_WORDLENGTH_8B;
    huart1.Init.StopBits = UART_STOPBITS_1;
    huart1.Init.Parity = UART_PARITY_NONE;
    huart1.Init.Mode = UART_MODE_TX_RX;
    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    HAL_UART_Init(&huart1);
}
```

 Copy code

Receive & Respond Example

```
C
```

```
uint8_t rx;
```

```
HAL_UART_Receive_IT(&huart1, &rx, 1);

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart->Instance == USART1)
    {
        if (rx == '1')
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET); // LED ON
        else if (rx == '0')
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET); // LED OFF

        HAL_UART_Receive_IT(&huart1, &rx, 1);
    }
}
```

 Copy code

 HC-05 sends raw ASCII bytes to STM32

3 Configure HC-05 (Optional but Recommended)

Enter AT mode (hold KEY while powering).

Common AT commands:

```
pgsql

AT+NAME=STM32_BT
AT+PSWD=1234
AT+UART=9600,0,0
AT+ROLE=0
```

 Copy code

4 Android App – Bluetooth Classic (SPP)

HC-05 uses Bluetooth Classic, not BLE.

Required Permissions (AndroidManifest.xml)

xml

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

<!-- Android 12+ -->
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.BLUETOOTH_SCAN" />
```

 Copy code

5 Android Code (Java – Simple & Stable)

Get Bluetooth Adapter

java

```
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

 Copy code

Connect to HC-05

java

```
BluetoothDevice device = bluetoothAdapter.getRemoteDevice("00:14:03:05:AA:BB");

UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"); // SPP
```

 Copy code

```
BluetoothSocket socket = device.createRfcommSocketToServiceRecord(uuid);
socket.connect();
```

Send Data to STM32

java

```
OutputStream outputStream = socket.getOutputStream();
outputStream.write("1".getBytes()); // LED ON
```

 Copy code

java

```
outputStream.write("0".getBytes()); // LED OFF
```

 Copy code

Receive Data from STM32

java

```
InputStream inputStream = socket.getInputStream();
byte[] buffer = new byte[1024];
int bytes = inputStream.read(buffer);
String received = new String(buffer, 0, bytes);
```

 Copy code

6 Simple Android UI

Buttons:

- ON → send "1"

- OFF → send "0"

TextView:

- Show received data
-

7 Data Protocol (Very Important)

Keep it simple:

Command	Meaning
1\n	LED ON
0\n	LED OFF
T\n	Request temperature
OK\n	STM32 response

Use \n to delimit messages.

8 Testing & Debugging

- ✓ Test HC-05 with **Bluetooth Terminal** app first
 - ✓ Check STM32 UART using **USB-TTL**
 - ✓ Match baud rate on all sides
 - ✓ Pair HC-05 before connecting
-

9 Recommended Tools

- Android Studio
 - Bluetooth Terminal HC-05
 - STM32CubeIDE
 - Logic analyzer (optional)
-

1 0 Want a Faster Way?

If you want:

- Kotlin version
- Full Android Studio project
- Custom protocol (JSON, CRC)
- Bidirectional streaming
- FreeRTOS on STM32

Tell me and I'll tailor it exactly to your setup 



how to find the hc05 in android?

To find an **HC-05 Bluetooth module on Android**, you must use **Bluetooth Classic discovery or bonded devices**, not BLE scanning.

Below is the **correct and reliable way**, step by step.

1 Make Sure HC-05 Is Discoverable

- ✓ Power ON the HC-05
- ✓ LED should **blink fast** (not slow)
- ✓ Default name: **HC-05**
- ✓ Default PIN: **1234 or 0000**

If the LED is blinking **slowly**, it's already connected to another device.

2 Pair HC-05 from Android Settings (Recommended)

📱 On your Android phone:

1. **Settings → Bluetooth**
2. Turn Bluetooth ON
3. Tap **Pair new device**
4. Select HC-05
5. Enter PIN 1234

✓ After pairing, HC-05 becomes a **bonded device**

This is the **easiest and most stable** method.

3 Find HC-05 in Android Code (Bonded Devices)

Java (Recommended)

java

```
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();  
  
Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();
```

```
for (BluetoothDevice device : pairedDevices) {  
    String name = device.getName();  
    String address = device.getAddress();  
  
    if (name != null && name.contains("HC")) {  
        Log.d("BT", "Found: " + name + " " + address);  
    }  
}
```

 Copy code

📌 This works only after pairing

4 Discover HC-05 Programmatically (Scan)

If not paired yet, use discovery.

Required Permissions (Android 12+)

xml

```
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"/>  
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>
```

 Copy code

Start Discovery

java

```
BluetoothAdapter adapter = BluetoothAdapter.getDefaultAdapter();  
  
if (!adapter.isDiscovering()) {  
    adapter.startDiscovery();  
}
```

 Copy code

Register Broadcast Receiver

java

```
BroadcastReceiver receiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        if (BluetoothDevice.ACTION_FOUND.equals(intent.getAction())) {
            BluetoothDevice device =
                intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);

            if (device != null) {
                Log.d("BT", device.getName() + " - " + device.getAddress());
            }
        }
    }
};
```

 Copy code

Register Receiver

java

```
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(receiver, filter);
```

 Copy code

 Discovery takes ~12 seconds

5 Connect to HC-05 After Finding It

Use SPP UUID:

java

```
UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

BluetoothSocket socket =
    device.createRfcommSocketToServiceRecord(uuid);

socket.connect();
```

 Copy code

6 Common Problems & Fixes

Problem	Fix
HC-05 not showing	Check LED blink (fast)
Scan finds nothing	Location permission required (Android ≤11)
Name is null	Use MAC address
Connection fails	Pair first in settings
Android 12 crash	Add BLUETOOTH_CONNECT permission

7 Best Practice (Strongly Recommended)

- ✓ Pair HC-05 in **Bluetooth Settings** first
- ✓ Use `getBondedDevices()` in app
- ✓ Avoid discovery unless necessary

Discovery is **slow and unreliable**.

8 Quick Debug Tip

Install “Bluetooth Terminal” app → connect → verify data TX/RX

If it works there, your Android code is correct.

If you want, I can:

- Write **Kotlin** version
- Auto-reconnect HC-05

Just tell me 



 Share  Add people ...

how to access in root mode in android?

“Root mode” on Android means getting **superuser (root) access**, similar to *administrator* on Linux.

 This is **not officially supported**, varies by device, and can **void warranty / break apps / brick the phone**.

Below is a **clear, realistic explanation** of what’s possible and how it’s usually done.

🔴 Important Reality Check

- You **cannot access root mode on a normal Android device by code alone**
- Root access requires **modifying the system** (bootloader + system image)
- Apps **cannot self-grant root** unless the device is already rooted

1 What “Root Access” Means

With root you can:

- ✓ Access /system , /data , /vendor
- ✓ Modify protected files
- ✓ Use commands like su
- ✓ Run privileged tools (iptables, tcpdump, system mods)

Without root:

- ✗ No access to system partitions
- ✗ No raw hardware access
- ✗ No kernel-level control

2 Check If Your Device Is Already Rooted

From Android app

Install Root Checker (Play Store)

From ADB

```
bash
```

```
adb shell
```

```
su
```

 Copy code

If prompt changes to # , you have root

If su: not found , not rooted

3 Standard Way to Get Root (Modern Android)

The only safe & modern method: Magisk

Magisk provides:

- Systemless root
 - Root hiding (important for banking apps)
 - Root permission control
-

4 General Rooting Process (High-Level)

 Exact steps depend on phone model (Samsung ≠ Pixel ≠ Xiaomi)

Step 1: Unlock Bootloader

bash

```
adb reboot bootloader  
fastboot oem unlock
```

 Copy code

 This wipes all data

Step 2: Patch Boot Image with Magisk

1. Extract `boot.img` from firmware
2. Install **Magisk app**
3. Patch `boot.img`
4. Flash patched image

bash

```
fastboot flash boot magisk_patched.img  
fastboot reboot
```

 Copy code

Step 3: Verify Root

```
bash  
  
adb shell  
su
```

 Copy code

You should get a **Magisk prompt**

5 Samsung Devices (Special Case)

Samsung does NOT support fastboot.

Requirements:

- Odin tool
- Unlock OEM in Developer Options
- Flash patched AP file
- Knox permanently tripped  (irreversible)

6 Android Emulator (Easy Root Access)

If you only need root for development/testing:

```
bash
```

```
adb root  
adb remount
```

 Copy code

-  Emulator supports root by default
 -  Real devices do not
-

7 Can You Root Programmatically or Temporarily?

-  No (anymore)

Old exploits (KingRoot, Framaroot):

- No longer work
 - Dangerous
 - Often malware
-

8 What Root Is Commonly Used For

- Custom ROMs
 - System automation
 - Packet sniffing
 - Hardware debugging
 - Reverse engineering
 - Full backups
 - Advanced Bluetooth/WiFi tools
-

9 Root vs What You Actually Need

If your goal is:

Goal	Root Needed?
HC-05 Bluetooth	✗ NO
UART / USB	✗ NO
File access	✗ NO
System apps	⚠ Sometimes
Kernel mods	✓ YES

👉 For HC-05 + STM32 communication, root is NOT required

10 Strong Advice

If you tell me:

- Phone model
- Android version
- Why you need root

I can tell you:

- ✓ If root is even necessary
- ✓ Safest method for your device
- ✓ Root-free alternative (often exists)

Most developers **don't actually need root**—they just think they do.

Ah! You're on Android 13. That actually makes things much clearer because **rooting Android 13 is significantly harder than older versions**, and in most cases, **you don't need root at all**—especially for things like connecting to an **HC-05 Bluetooth module**. Let me break it down carefully.

1 Root on Android 13 – Reality Check

- Android 13 has **stronger security (SEAndroid + dm-verity)**
 - **Apps cannot gain root automatically**
 - Only way to get root: **unlock bootloader + patch boot image with Magisk**
 - Root can **break some apps, void warranty, and disable OTA updates**
 - Most **Bluetooth / UART communication apps do NOT require root**
-

2 Rooting Options for Android 13

Magisk (Systemless Root)

- Currently the **best method**
- Works on most phones with **unlocked bootloader**
- Keeps system partition unmodified
- Can hide root from apps

Steps (general outline):

1. Unlock bootloader (Developer Options → OEM Unlock)
2. Extract stock boot image

3. Patch boot image using Magisk app
4. Flash patched boot image via fastboot
5. Reboot → root access

Samsung / locked-bootloader devices

- Need Odin / custom recovery (TWRP)
 - Knox is permanently tripped
 - More complicated and risky
-

3 Android 13 Emulator – Easy Root

If you just want to test an app, you can use an emulator:

bash

```
adb root  
adb remount
```

 Copy code

- Emulator has root by default
 - Safe, no warranty issues
 - Can test Bluetooth simulation (with limitations)
-

4 Important: Do You Actually Need Root for HC-05?

No. Absolutely not.

All you need for Android 13:

- Bluetooth Classic permissions
- BLUETOOTH_CONNECT , BLUETOOTH_SCAN permissions

- Pairing with HC-05
- BluetoothSocket SPP connection

Root does not give you anything extra for communicating with STM32 over HC-05.

5 Recommendation for Android 13

For HC-05 + STM32F407 app:

- Do not root your phone
 - Use bonded devices API (`getBondedDevices()`)
 - Scan for unpaired devices if necessary
 - Connect using UUID `00001101-0000-1000-8000-00805F9B34FB`
- Safe, stable, works on all modern phones
-

If you want, I can give you a **full working Android 13 Kotlin example** that:

- Scans for HC-05
- Shows devices in a list
- Connects via SPP
- Sends and receives data to STM32

This would **work without root at all**.

Do you want me to do that?



if I want to use rf module to send data from android to hc05 to other rf receiver?

Good question — this is a **very common point of confusion**, so let's clear it up cleanly.

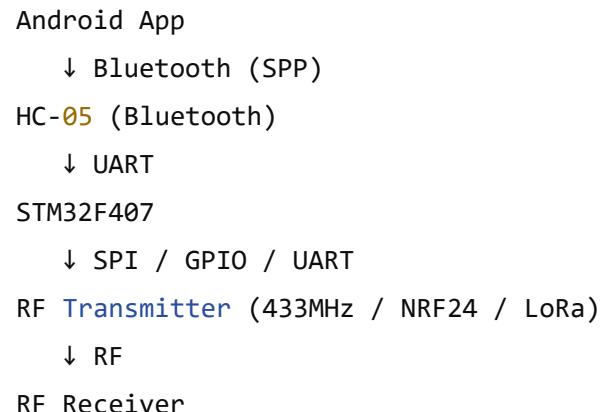
Short answer first:

- ✗ **Android cannot directly transmit RF data to an RF module**
- ✓ You must use **Bluetooth (HC-05) → MCU → RF transmitter → RF receiver**

Android **never talks to RF hardware directly**. It always talks to Bluetooth/Wi-Fi/USB, and your microcontroller acts as the bridge.

Correct Architecture (This WILL Work)

java



↓
Target MCU / Device

- 👉 HC-05 is ONLY a Bluetooth cable replacement
 - 👉 STM32 handles RF logic
-

✗ What Is NOT Possible

Idea	Possible?	Why
Android → RF module directly	✗	No RF hardware access
Android controlling RF registers	✗	Kernel-level access required
Root enabling RF TX	✗	Still no RF driver
HC-05 forwarding RF itself	✗	HC-05 is Bluetooth-only

1 Choose Your RF Module

Common RF modules used with STM32:

RF Module	Interface	Notes
433 MHz ASK/OOK	GPIO	Very simple, no ACK
NRF24L01	SPI	Fast, bidirectional
LoRa (SX127x)	SPI	Long range

RF Module	Interface	Notes
HC-12	UART	Very easy, transparent

💡 HC-12 is easiest if you want Bluetooth-like behavior over RF.

2 STM32 Role (Bridge Controller)

STM32:

- Receives commands from Android via HC-05
 - Encodes data
 - Sends via RF module
 - Receives RF data
 - Sends response back to Android
-

3 Data Flow Example

Android → STM32

text

"LED_ON\n"

 Copy code

STM32 → RF

text

0xA5 0x01 0x01 0x5A // example packet

 Copy code

RF Receiver → MCU

text

ACK_OK

 Copy code

STM32 → Android

text

"OK\n"

 Copy code

4 STM32 Firmware Example (Pseudo)

C

```
void processBluetoothCommand(char *cmd)
{
    if (strcmp(cmd, "LED_ON") == 0)
        rf_send(0x01);
    else if (strcmp(cmd, "LED_OFF") == 0)
        rf_send(0x00);
}
```

 Copy code

5 Android App Code (Same as HC-05 Only)

Your Android app **does not change at all**.

java

 Copy code

```
outputStream.write("LED_ON\n".getBytes());
```

Android:

- Doesn't know RF exists
 - Doesn't care which RF module you use
-

6 If You Want Two-Way RF Communication

STM32:

- Listens to RF receiver
- Forwards data back to Android

text

Android ← HC-05 ← STM32 ← RF receiver

 Copy code

7 Best RF Module Choices (Recommendation)

HC-12 (UART RF)

- UART like HC-05
- Easy firmware
- 1 km range
- Very stable

NRF24L01

- High speed
- Auto ACK

- Needs SPI driver

3 433 MHz

- Cheapest
 - No reliability
 - Needs encoding
-

8 Key Takeaway (Important)

- Android never sends RF data
- STM32 always handles RF
- Android only sends Bluetooth commands

This architecture is **industry standard** and scalable.

9 Want Me to Design It Fully?

If you tell me:

- Which RF module you plan to use
- One-way or two-way communication
- Range requirement
- Number of nodes

I can give you:

- ✓ Full STM32 code
- ✓ Android protocol
- ✓ RF packet format
- ✓ Wiring diagram

Just tell me 



if baud rate is low, the distance will be longer?

Short answer: sometimes yes, but not the way most people think.

Let's be precise 

General Rule (for RF & serial radios)

Lower baud rate → better reliability → longer usable range

But:

- Baud rate does NOT increase transmit power
- It improves receiver sensitivity and noise tolerance

So the maximum physical range stays the same, but the working range increases.

Why Lower Baud Rate Helps

Receiver Sensitivity

At lower baud rates:

- Symbols are longer
- Receiver can average noise better

- Lower **SNR** (Signal-to-Noise Ratio) is acceptable

Result:

Weak signals become readable at longer distances

2 Narrower Bandwidth

Lower baud rate → narrower bandwidth:

- Less noise enters receiver
- Better selectivity

This is why **LoRa** at 300 bps can go kilometers.

3 Error Probability

Higher baud rate:

- Bits are closer together
- More errors due to noise

Lower baud rate:

- Fewer bit errors
 - Fewer retries
 - Effective range increases
-



Examples by RF Type

- ◆ HC-12 (UART RF)

Baud Rate	Typical Range
9600	~500 m
1200	~1000 m
300	~1800 m

(Line of sight)

◆ LoRa (SX127x)

Data Rate	Range
37.5 kbps	~2 km
4.8 kbps	~10 km
300 bps	15–20 km

◆ NRF24L01

Air Rate	Range
2 Mbps	~50 m
250 kbps	~1000 m

- ◆ 433 MHz ASK

Lower data rate = dramatically better range

✖ When Lower Baud Rate Does NOT Help

- Antenna is bad
- RF power too low
- Interference is extreme
- Hardware is poor quality

Baud rate cannot fix:

- Bad antennas
 - Wrong impedance
 - Low TX power
-

💡 Important Distinction (Very Important)

Term	Meaning
UART baud rate	MCU ↔ module speed
RF data rate	Actual air speed

⚠ These are NOT always the same

Example:

- UART = 9600
- RF = 250 kbps (NRF24)

Changing UART baud alone may **not** change RF speed.

Practical Advice

If you want longer range:

1. Lower RF air data rate
 2. Use better antennas
 3. Increase TX power (within legal limits)
 4. Choose lower frequency (433 MHz > 2.4 GHz)
 5. Use CRC / ACK / retries
-

Answer in One Line

-  Lower baud rate improves usable range and reliability,
 -  but it does not magically increase transmit power.
-

If you tell me:

- Which RF module you're using
- Current baud rate
- Required distance

I can give you **exact settings** to maximize range 



I mean, hc05

Ah! Now I understand — you specifically mean **HC-05 Bluetooth module**. Let's talk about **baud rate vs range for HC-05**.

1 How HC-05 Works

- HC-05 is a **Bluetooth Classic (2.0 / SPP) module**
 - Bluetooth Classic range is mostly determined by:
 - Transmit power class (Class 1, 2, 3)
 - Antenna quality
 - Environment / obstacles
 - UART baud rate (between STM32 ↔ HC-05) is **internal to the module**. It does not directly affect RF range.
-

2 Can Lower UART Baud Rate Increase HC-05 Range?

- No direct effect on RF range
- UART baud rate = speed of serial communication between STM32 and HC-05
- Lowering it can reduce chance of data loss if wiring is noisy or MCU is slow
- But over the air (Bluetooth) range stays the same

Example:

- “HC-05 TX/RX UART at 9600 bps or 115200 bps”
- “Bluetooth still transmits at ~2–3 Mbps air rate”
- “Max range is still ~10 m (Class 2) or 100 m (Class 1)”

3 What Can Affect HC-05 Range

Factor	Effect
Bluetooth class	Class 1 → 100 m, Class 2 → 10 m
Antenna quality	Poor soldered wire → lower range
Environment	Walls, metal, interference reduce range
Power supply	Low voltage → reduced TX power

4 Why People Think “Lower Baud → Longer Range”

- Sometimes people lower UART baud to avoid lost bytes due to MCU timing issues
- They confuse **stable** UART communication with RF range
- On HC-05, **UART baud rate only affects reliability of MCU ↔ HC-05**, not Bluetooth distance

5 HC-05 Max Range Tips

- Use **Class 1 module** (10x stronger than default Class 2)
- Keep **line of sight** between devices
- Avoid interference (Wi-Fi 2.4 GHz, microwaves)
- Ensure **good antenna and 3.3–5V stable supply**

 Bottom line for HC-05:

Lowering UART baud rate does not make Bluetooth travel further. It only makes STM32 ↔ HC-05 communication more stable.

Actual Bluetooth range is determined by module class, antenna, and environment.

If you want, I can make a [table showing HC-05 classes, recommended baud rates, and practical max ranges](#) for STM32 projects — very handy for planning wireless control.

Do you want me to do that?



[how to set class in hc05](#)

Yes! You [can change the Bluetooth “class”](#) (power/range) of an HC-05 module using [AT commands](#). I'll explain exactly how.

1 HC-05 Classes (Transmit Power)

Class	Typical TX Power	Approx. Range (line of sight)
Class 1	100 mW	~100 m
Class 2	2.5 mW	~10 m
Class 3	1 mW	~1 m

Default HC-05 modules are usually Class 2 (10 m range).

2 Enter AT Mode

Steps:

1. Power off HC-05
 2. Hold **KEY / EN** pin HIGH while powering on → LED blinks slowly
 3. HC-05 is now in **AT command mode**
 - LED blinks **slowly** (~2s) instead of fast
 4. Connect HC-05 to STM32 or USB-TTL adapter (RX/TX 3.3–5V)
-

3 AT Command Communication

Baud rate in AT mode is usually 38400 bps.

Test connection:

text

AT

 Copy code

Response:

text

OK

 Copy code

4 Set Bluetooth Class

Command format:

text

AT+CLASS=<hex_value>

 Copy code

Class values (hex):

Class	Hex	Description
Class 1	0x000104	Maximum range (100 m)
Class 2	0x000200	Default (10 m)
Class 3	0x000400	Minimal (1 m)

Example: Set to Class 1

text

AT+CLASS=000104

 Copy code

Response:

text

OK

 Copy code

5 Save Settings

Some modules save automatically; some need:

text

Then power cycle.

6 Optional Useful AT Commands

Command	Use
AT+NAME=STM32_BT	Set Bluetooth device name
AT+UART=9600,0,0	Set UART baud, stop bits, parity
AT+ROLE=0	Set as slave (default)
AT+PSWD=1234	Set pairing PIN

7 Notes / Warnings

- Class 1 modules consume more power → 100 mW
 - Legal limits: Check your country's RF regulations
 - UART baud rate in AT mode is not the same as data mode
-

Summary

1. Enter AT mode (KEY high on power-up)
2. Connect to HC-05 via UART at 38400 bps
3. Send AT+CLASS=000104 to set Class 1
4. Reset module (AT+RESET)

5. HC-05 now has maximum range (~100 m)

If you want, I can **write a ready-to-use STM32 HAL code snippet to send AT commands to HC-05 and change class** — so you can do it directly from your microcontroller without a PC.

Do you want me to do that?

