### [**SPP API ESP-IDF WEBLINK**](https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/bluetooth/esp_spp.html)

### **What is Bluetooth SPP (Serial Port Profile)?**

Bluetooth **SPP (Serial Port Profile)** emulates a serial cable (RS-232) over Bluetooth. It allows two Bluetooth devices to communicate over a virtual serial port — meaning data can be sent and received just like with traditional UART/RS232, but wirelessly.

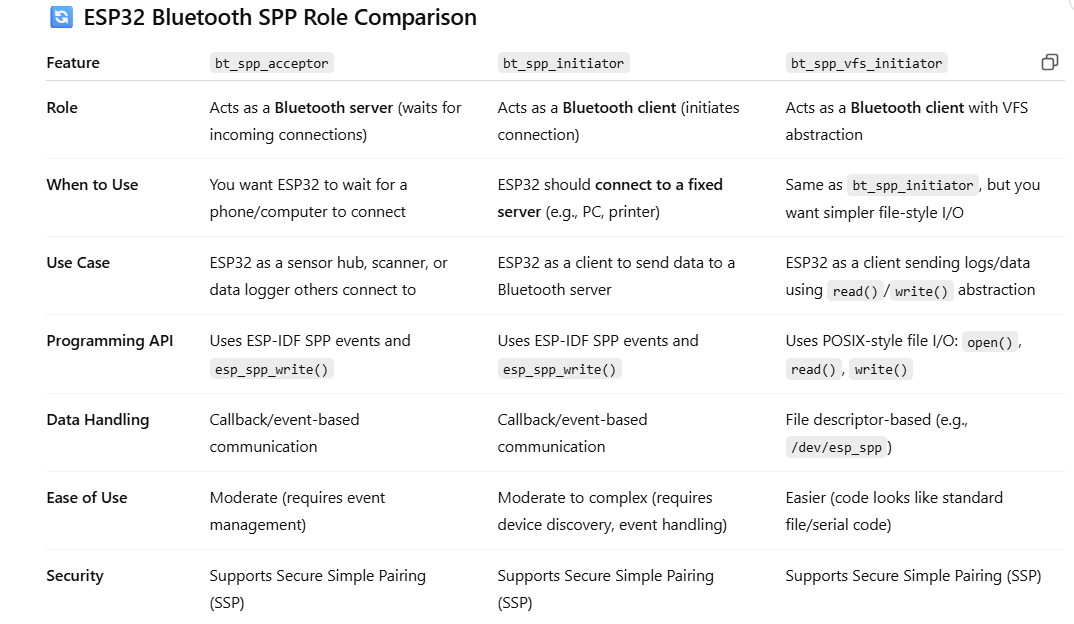
SPP is widely used for:

* Sending sensor data to a smartphone or computer.
* Serial communication between embedded devices and PCs.
* Bluetooth-based console/debug logs from microcontrollers.

The **SPP API** in ESP-IDF allows you to implement Bluetooth SPP functionality on the ESP32 using the Bluedroid stack (classic Bluetooth).

There are **two main roles**:

1. **SPP Acceptor** (Server): Waits for connections.
2. **SPP Initiator** (Client): Starts the connection.



We want ESP32 to wait for a phone to connect. Hence, bt\_spp\_acceptor must be used.

[BT\_SPP\_ACCEPTOR GITHUB CODELINK](https://github.com/espressif/esp-idf/tree/v5.4.1/examples/bluetooth/bluedroid/classic_bt/bt_spp_acceptor)

### **How It Works (Step by Step)**

1. **Start Bluetooth**
   * Prepares storage (NVS) and sets up Classic Bluetooth (not BLE).
   * Releases any unused Bluetooth Low Energy memory.
   * Initializes and turns on the Bluetooth controller.
   * Initializes the Bluetooth stack (Bluedroid).
2. **Register Event Handlers**
   * Two types of callbacks are set up:  
     + **GAP callback** for handling pairing and connection settings.
     + **SPP callback** for handling data and connection events.
3. **Set Up as a Server**
   * When Bluetooth is ready, the ESP32:  
     + Starts an SPP server named "SPP\_SERVER".
     + Sets its Bluetooth name (CONFIG\_EXAMPLE\_LOCAL\_DEVICE\_NAME).
     + Makes itself visible and connectable to other devices.
4. **Wait for Connection**
   * Once a phone/computer connects:  
     + The ESP32 logs the connection details.
     + It starts tracking the time to calculate data speed.
5. **Receive Data**
   * When data is received:  
     + If in **data mode**, it will print the data (if small).
     + If in **speed mode**, it calculates and displays the transmission speed every 3 seconds.
6. **Pairing/Security**
   * If pairing is required:  
     + It supports PIN pairing (default "1234" or 16-digit code).
     + Secure Simple Pairing (SSP) is also supported if enabled.

Any change in buffer register=> interrupt  
Shift register starts sending data

esp\_spp\_cb(...) is the **Bluetooth SPP event handler callback**. It listens for **SPP (Serial Port Profile)** events such as:

ESP\_SPP\_DATA\_IND\_EVT — Incoming Data Received