ChatGPT

# Waveshare ST-Series Serial Servo Control

**Supported Hardware & Wiring:** Waveshare's "Servo Driver with ESP32" board integrates an ESP32, an OLED, and two 3-pin bus-servo ports (internally tied). Power the board with a DC 6–12 V supply matching your ST servo voltage (e.g. 12 V for ST3215) [1] [2]. Plug the ST-series servo into either 3-pin port (they're paralleled) and common-ground with the ESP32. For the USB adapter ("Bus Servo Adapter (A)"), connect the adapter's servo-bus output to the servo's bus line and a suitable servo power supply; the adapter itself is USB-powered. In either case, connect TX/RX properly: e.g. when using an Arduino/ESP32 board as host, wire Arduino TX→adapter "RX", Arduino RX→adapter "TX", and ground to ground [3]. On the ESP32 driver board, the UART for servos defaults to GPIO18 (RX) and GPIO19 (TX) [4] [5].

**Libraries & Firmware:** Use the Arduino IDE (with ESP32 core if needed). Install Waveshare's open-source **SCServo** library (by copying the library folder into `Arduino/libraries`) for bus-servo commands [6] [7]. For the ESP32 driver board's OLED functionality, also install Adafruit SSD1306 and NeoPixel libraries via Library Manager [6]. Waveshare provides example firmware: see the **Servo-Driver-with-ESP32** GitHub repo (contains `ServoDriver.ino` and SCServo files) [7] [8]. For the USB adapter on PC, one can use the same SCServo library in an Arduino sketch (using a USB-serial module) or PC code at 1 Mbps.

**Serial Parameters:** The ST-series servo bus uses 1,000,000 baud, 8 data bits, no parity, 1 stop bit (8N1) [9] [5]. In Arduino/ESP32, begin the bus UART at 1e6 bps. For example on ESP32: `Serial1.begin(1000000, SERIAL_8N1, S_RXD, S_TXD);` where `S_RXD=18` and `S_TXD=19` by default [5]. Then assign the SCServo object's port pointer, e.g. `st.pSerial = &Serial1;` (or `&Serial` on an Arduino UNO) [9] [5]. Common pitfalls: ensure TX/RX pins match the wiring, and always set the bus speed to 1,000,000 baud.

**Arduino Code Example (ST servo, e.g. ST3020/ST3215):** The SCServo library provides high-level calls. Example (Arduino/USB-adapter or ESP32) to move servo ID 1 to center (2047 of 0–4095 range, at speed 1500, accel 50):

```
#include <SCServo.h>
SMS_STS st;
#define S_RXD 18
#define S_TXD 19

void setup() {
    Serial1.begin(1000000, SERIAL_8N1, S_RXD, S_TXD); // bus UART
    st.pSerial = &Serial1;
    delay(1000);
}
void loop() {
    st.WritePosEx(1, 2047, 1500, 50);  // position=2047 (center), speed=1500,
accel=50
```

```
    delay(2000);
}
```

This is adapted from known examples [10] [11]. On an Arduino UNO, replace with `Serial.begin(1000000); st.pSerial=&Serial;` (UNO has only one UART) [9]. The `WritePosEx(id,pos,spd,acc)` call sends the move command on the bus; here it centers servo 1 at midrange.

**EEPROM-Mode Sequence:** To change servo configuration (ID or mode), use the SCServo EEPROM unlock/write/lock calls. For example, to set servo 1's ID to 9:

```
st.unLockEprom(1);                  // allow EEPROM write
st.writeByte(1, SMS_STS_ID, 9);    // write new ID (register SMS_STS_ID)
st.LockEprom(9);                   // lock EEPROM at new ID
```

This pattern (unlock, write, lock) is required for safe parameter changes [12]. Similarly, to set "servo mode" vs "motor mode", write to the SMS_STS_MODE register (check your servo's manual). After programming, always lock with `LockEprom(id)`. A common pitfall is forgetting to unlock before writing or using the wrong register.

**Troubleshooting & Tips:** - **Power & Voltage:** Verify your servo's supply voltage (ST series: 6–12 V) matches the adapter/driver supply [1]. Under-powering or wrong polarity will prevent motion.
- **ID Conflicts:** On first use, all servos default to ID 1. If multiple servos share ID, commands may not take effect. Scan for active IDs using `st.Ping(id)` in a loop [13]. Example:

```
for (int id=1; id<=10; id++) {
    if (st.Ping(id) != -1) Serial.println(id);
}
```

This finds connected servos. Then address each by its unique ID.
- **Mode & Calibration:** Ensure the servo is in **"servo mode"** (absolute positioning) not continuous motor mode. If needed, send the appropriate mode byte (see Waveshare/ST servo docs) using EEPROM write. Some servos require a zero-offset calibration; Waveshare's example calls `st.CalibrationOfs(id)` (after setting center position physically) to record the mid-point.
- **Library Installation:** After copying SCServo library into Arduino's libraries folder [6] [7], restart the IDE. Include `<SCServo.h>` in your sketch.
- **Serial Parameters:** Always use 1,000,000 baud. Using USB-Serial adapters, double-check that no USB-serial auto-baud or mismatched settings are active.

**Sources:** Official Waveshare documentation and examples [6] [7] (including their Arduino demos), plus community guides [3] [5] and user projects [10] [11]. These resources provide tested code and sequences for SC/ST series servos. By following the wiring above and using the SCServo library calls, an ST-series servo can be reliably commanded to center position.

---

1 download.kamami.pl
https://download.kamami.pl/p1181056-ST3215_Servo_User_Manual.pdf

2 7 Servo Driver with ESP32 User Guide
https://spotpear.com/index/study/detail/id/885.html

3 9 10 12 Operating a Serial Servo (Waveshare) Using Arduino - Instructables
https://www.instructables.com/Operating-a-Serial-Servo-Waveshare-Using-Arduino/

4 61ac6eb739d1c4b9c70a2cb01c...
https://gitlab.svfactory.com/yanpeng.luo/st3215_close_or_open_control/-/blob/61ac6eb739d1c4b9c70a2cb01c0f8c1d6a79fa48/SCServo/examples/STSCL/SyncWritePos/SyncWritePos.ino

5 11 13 WaveShare servo Driver - Motors, Mechanics, Power and CNC - Arduino Forum
https://forum.arduino.cc/t/waveshare-servo-driver/1291256

6 files.waveshare.com
https://files.waveshare.com/upload/d/d4/Servo_Driver_with_ESP32_User_Manual.pdf

8 GitHub - waveshare/Servo-Driver-with-ESP32: The Web app example for Servo Driver with ESP32.
https://github.com/waveshare/Servo-Driver-with-ESP32