

JSON Command Communication Guide

This document explains how to use **JSON-based commands** to communicate with the robot driver board.

It serves as a reference for developers who wish to control robot functions, extend firmware capabilities, or integrate the system with higher-level applications such as Python scripts, ROS2 nodes, or Web control interfaces.

1. Introduction

All Lygion robot driver boards, including **Robot Driver with ESP32S3 Lite (A)**, use a unified JSON command format for communication between the **upper controller (host PC, Web app, ROS2, etc.)** and the **lower controller (ESP32S3)**.

This allows easy debugging, rapid development, and smooth cross-platform integration.

2. Why JSON?

Using JSON as the communication protocol provides several major advantages:

✓ Clear Structure & Readability

JSON uses key-value pairs, making data self-explanatory:

```
{"T":11,"id":1,"pos":2047,"spd":0,"acc":0}
```

- Easy to read for both humans and machines
 - No need for custom binary parsing or delimiters
 - Ideal for debugging and logging
-

✓ Cross-Platform Compatibility

Every modern programming environment supports JSON parsing — including **C/C++**, **Python**, **JavaScript**, and **ROS2**.

This means you can control the robot directly from:

- A **Web App** using WebSocket or HTTP
 - A **Python script** via USB CDC
 - A **ROS2 node** publishing JSON messages
-

✓ Extensible and Backward Compatible

Adding new parameters or commands won't break existing code.

For example:

```
{"T":21,"id":1,"pos":2047,"spd":0,"acc":0,"cl":500}
```

Older firmware simply ignores unknown fields, ensuring long-term compatibility.

✓ Works with Modern Communication Methods

JSON commands can be sent through:

- **USB CDC**
- **WebSocket**
- **HTTP**

- **ESP-NOW**

Perfect for integration with cloud dashboards, web-based control panels, and IoT applications.

✓ Easy Debugging & Logging

- Directly send JSON commands using serial terminal or web console
- Log and replay commands easily during development
- System feedback is also JSON-based for consistency:

```
{"status":"ok","pos":[120,80,45],"voltage":11.8}
```

3. Getting Started

Web Interface

In the Web control panel, the lower half of the page shows the **JSON Command List**.

Click on a command to auto-fill it into the **JSON Interface** input box.

You can edit the JSON before pressing **SEND** to execute it.

💡 Note:

“Automation Scripts” is for multiple commands executed in sequence, while “JSON Interface” is for single, on-demand commands.

Host Communication Options

You can also communicate with the driver board through:

- **USB-CDC Serial** (recommended for local testing)
- **WebSocket / HTTP** (for network applications)
- **ESP-NOW** (for wireless peer-to-peer control)

Detailed examples will be provided in later sections.

4. Command Reference

System Control

Command	JSON Example	Description
CMD_BREAK_LOOP	<code>{"T":0}</code>	Stop current task execution
CMD_ESP32_REBOOT	<code>{"T":600}</code>	Reboot the device
CMD_CLEAR_NVS	<code>{"T":601}</code>	Clear NVS storage (useful when Wi-Fi/ESP-NOW malfunctions)
CMD_RESET	<code>{"T":602}</code>	Format filesystem and reset the device
CMD_SET_MSG_OUTPUT	<code>{"T":604,"echo":1,"uart":0,"usb":1}</code>	Configure debug message output channels
CMD_SERIAL_FORWARDING	<code>{"T":605,"sf":1}</code>	Enable serial forwarding to use external servo debugging tools

Servo Control Commands

ST/SM Servo – STS/SMS Series

Command	JSON Example	Description
CMD_STSM_CTRL	<code>{"T":11,"id":1,"pos":2047,"spd":0,"acc":0}</code>	Move servo to position <code>pos</code> with speed <code>spd</code> and acceleration <code>acc</code>

Command	JSON Example	Description
CMD_STSM_SET_MIDDLE	<code>{"T":12,"id":1}</code>	Set current position as middle
CMD_STSM_CHANGE_ID	<code>{"T":13,"old_id":1,"new_id":2}</code>	Change servo ID
CMD_STSM_TORQUE_LOCK	<code>{"T":14,"id":1,"state":1}</code>	Enable/disable torque lock (1 = lock)
CMD_STSM_FEEDBACK	<code>{"T":15,"id":1}</code>	Read servo feedback

HL Servo – HLS Series

Command	JSON Example	Description
CMD_HL_CTRL	<code>{"T":21,"id":1,"pos":2047,"spd":0,"acc":0,"cl":500}</code>	Move servo with speed, acceleration, and current limit
CMD_HL_SET_MIDDLE	<code>{"T":22,"id":1}</code>	Set current position as middle
CMD_HL_CHANGE_ID	<code>{"T":23,"old_id":1,"new_id":2}</code>	Change servo ID
CMD_HL_TORQUE_LOCK	<code>{"T":24,"id":1,"state":1}</code>	Enable/disable torque lock
CMD_HL_FEEDBACK	<code>{"T":25,"id":1}</code>	Read servo feedback

SC Servo – SCS Series

Command	JSON Example	Description
CMD_SC_CTRL	<code>{"T":31,"id":1,"pos":511,"time":0,"spd":0}</code>	Move servo with specified position/time/speed
CMD_SC_CHANGE_ID	<code>{"T":33,"old_id":1,"new_id":2}</code>	Change servo ID
CMD_SC_TORQUE_LOCK	<code>{"T":34,"id":1,"state":1}</code>	Enable/disable torque lock
CMD_SC_FEEDBACK	<code>{"T":35,"id":1}</code>	Read servo feedback

Display & Delay

Command	JSON Example	Description
CMD_DELAY	<code>{"T":51,"delay":1000}</code>	Delay in milliseconds
CMD_DISPLAY_SINGLE	<code>{"T":202,"line":1,"text":"Hello, world!","update":1}</code>	Display text on specific OLED line
CMD_DISPLAY_UPDATE	<code>{"T":203}</code>	Refresh display content
CMD_DISPLAY_FRAME	<code>{"T":204,"l1":"Hello!","l2":"world!","l3":"Hello!","l4":"world!"}</code>	Display multiple lines at once
CMD_DISPLAY_CLEAR	<code>{"T":205}</code>	Clear OLED display
CMD_BUZZER_CTRL	<code>{"T":206,"freq":1000,"duration":1000}</code>	Control buzzer frequency and duration

Wireless Configuration

Command	JSON Example	D
CMD_SET_WIFI_MODE	<code>{"T":400,"mode":1,"ap_ssid":"Robot","ap_password":"12345678","channel":1,"sta_ssid":"","sta_password":"","password"}</code>	C A
CMD_SET_ESP_NOW_MODE	<code>{"T":411,"mode":1}</code>	E d N r
CMD_ADD_MAC	<code>{"T":414,"mac":"FF:FF:FF:FF:FF:FF"}</code>	A a li
CMD_ESP_NOW_SEND	<code>{"T":413,"mac":"FF,FF,FF,FF,FF,FF","data":"{"T":205,"freq":500,"duration":30}"}</code>	S c p N

File System & Automation

Command	JSON Example	Description
CMD_CREATE_MISSION	<code>{"T":301,"name":"mission1","intro":"Mission file introduction"}</code>	Create a new mission file
CMD_APPEND_STEP_JSON	<code>{"T":303,"name":"boot","json":{"T":205,"freq":500,"duration":30}}</code>	Append a JSON command to a mission file
CMD_INSERT_STEP_JSON	<code>{"T":304,"name":"boot","step":2,"json":{"T":205,"freq":500,"duration":30}}</code>	Insert a command at a specific step
CMD_REPLACE_STEP_JSON	<code>{"T":305,"name":"boot","step":2,"json":{"T":205,"freq":500,"duration":30}}</code>	Replace a command in a mission file
CMD_DELETE_STEP	<code>{"T":306,"name":"boot","step":2}</code>	Delete a specific command from mission file
CMD_RUN_STEP	<code>{"T":307,"name":"boot","step":2}</code>	Execute a specific step
CMD_RUN_MISSION	<code>{"T":308,"name":"boot","interval":1000,"loop":1}</code>	Run a mission file; loop=-1 for infinite repeat
CMD_DELETE_MISSION	<code>{"T":309,"name":"boot"}</code>	Delete a mission file

5. Development Tips

- Always use **double quotes (")** in JSON keys and strings.
- For commands containing nested JSON (like `CMD_ESP_NOW_SEND`), use **escape characters (\)** before each double quote.
- Test commands first via **Web Interface** before embedding them into automation scripts.
- Avoid excessive command frequency — use `CMD_DELAY` when needed between movements.

6. Example: Servo Control via Python

```
import serial, json, time

ser = serial.Serial('COM5', 115200)

cmd = {"T":11,"id":1,"pos":2047,"spd":0,"acc":0}
ser.write((json.dumps(cmd) + '\n').encode())

time.sleep(1)
print("Servo moved to center position.")
```

7. Summary

Using JSON commands makes robot control **simple, extensible, and language-independent**.

Whether you are building a Web dashboard, ROS2 controller, or IoT interface, the same command structure applies across platforms.

This approach greatly simplifies multi-device communication and helps developers build reliable robotic systems faster.