

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

## LinkArm Ctrl – Mechanical Arm Control Commands

### Joint & Kinematics Control

Command	JSON Example	Description
<b>CMD_LINK_ARM_SC_JOINTS_CTRL_ANGLE</b>	<code>{"T":134,"ang":[0,0,0,0]}</code>	Control each joint of the arm in <b>degrees</b> . The four <code>ang[]</code> values correspond to servos <b>ID31, ID32, ID33, ID34</b> .
<b>CMD_LINK_ARM_SC_JOINTS_CTRL_RAD</b>	<code>{"T":135,"rad":[0,0,0,0]}</code>	Control each joint of the arm in <b>radians</b> . The four <code>rad[]</code> values correspond to servos <b>ID31, ID32, ID33, ID34</b> .
<b>CMD_XYZG_CTRL</b>	<code>{"T":136,"xyzg":[260.5,0,122.38,50]}</code>	Cartesian control with inverse kinematics. Moves the arm directly to the target point <b>without interpolation</b> . Parameters: <code>x, y, z, g</code> where <code>g=0</code> = fully closed gripper, <code>g=50</code> = fully open. Recommended for host-side interpolation.
<b>CMD_FPV_ABS_CTRL</b>	<code>{"T":137,"rbzg":[260.5,0,122.38,50]}</code>	FPV-style absolute control with IK for vertical-plane motion. Parameters: <code>r</code> (horizontal reach, $\geq 0$ ), <code>b</code> (base rotation, +right / -left), <code>z</code> (height), <code>g</code> (gripper angle: 0 closed, 50 open). No interpolation.
<b>CMD_SMOOTH_XYZG_CTRL</b>	<code>{"T":138,"xyzg": [236.5,0,122.38,0],"spd":0.4}</code>	Smooth Cartesian motion with interpolation and soft-start/soft-stop. Ensures the end-effector moves along a <b>straight path</b> . This command is <b>blocking</b> , depending on travel distance.
<b>CMD_SMOOTH_FPV_ABS_CTRL</b>	<code>{"T":139,"rbzg": [236.5,0,122.38,0],"spd":0.4,"br":200}</code>	Smooth FPV motion with interpolation and soft start/stop. Blocking behavior depends on motion distance.
<b>CMD_SET_LINK_ARM_FEEDBACK_FLAG</b>	<code>{"T":145,"flag":0,"hz":10}</code>	Enable/disable USB CDC feedback for joint angles & load data. <code>flag=0</code> disable, <code>flag=1</code> enable. <code>hz</code> : feedback frequency.

## Delay & Display – Delay / OLED Display / Buzzer

### Utility & UI Output

Command	JSON Example	Description
<b>CMD_DELAY</b>	<code>{"T":51,"delay":1000}</code>	Delay execution (ms).
<b>CMD_SET_SINGLE_COLOR</b>	<code>{"T":200,"id":40,"set":[1,255,0,0]}</code>	Set LED color on the node board (default ID = 40). First value = LED index (1 or 2), followed by <b>R G B</b> (0–255, 8-level quantized).
<b>CMD_SET_ALL_COLOR</b>	<code>{"T":201,"id":40,"set":[9,0,0]}</code>	Set both LEDs on the node board to the same color. RGB values follow the same 8-level quantization.
<b>CMD_DISPLAY_SINGLE</b>	<code>{"T":202,"line":1,"text":"Hello, world!","update":1}</code>	Display text on a specific line of the OLED screen. If <code>update=0</code> , multiple lines can be set before calling <b>CMD_DISPLAY_UPDATE</b> .
<b>CMD_DISPLAY_UPDATE</b>	<code>{"T":203}</code>	Refresh the OLED display.
<b>CMD_DISPLAY_FRAME</b>	<code>{"T":204,"l1":"Hello!","l2":"world!","l3":"Hello!","l4":"world!"}</code>	Set all display lines at once.
<b>CMD_DISPLAY_CLEAR</b>	<code>{"T":205}</code>	Clear the OLED screen.
<b>CMD_BUZZER_CTRL</b>	<code>{"T":206,"freq":1000,"duration":1000}</code>	Control the buzzer. <code>freq</code> in Hz, <code>duration</code> in ms.

### SC Servo – SCS Series

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

Command	JSON Example	Description
<b>CMD_SC_FEEDBACK</b>	<code>{"T":35,"id":1}</code>	Read servo feedback

## Wireless Configuration

Command	JSON Example	
<b>CMD_SET_WIFI_MODE</b>	<code>{"T":400,"mode":1,"ap_ssid":"Robot","ap_password":"12345678","channel":1,"sta_ssid":"ssid","sta_password":"password"}</code>	D C A
<b>CMD_SET_ESP_NOW_MODE</b>	<code>{"T":411,"mode":1}</code>	E d N r
<b>CMD_ADD_MAC</b>	<code>{"T":414,"mac":"FF:FF:FF:FF:FF:FF"}</code>	A a li
<b>CMD_ESP_NOW_SEND</b>	<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
<b>CMD_CREATE_MISSION</b>	<code>{"T":301,"name":"mission1","intro":"Mission file introduction"}</code>	Create a new mission file
<b>CMD_APPEND_STEP_JSON</b>	<code>{"T":303,"name":"boot","json":{"T":205,"freq":500,"duration":30}}</code>	Append a JSON command to a mission file
<b>CMD_INSERT_STEP_JSON</b>	<code>{"T":304,"name":"boot","step":2,"json":{"T":205,"freq":500,"duration":30}}</code>	Insert a command at a specific step
<b>CMD_REPLACE_STEP_JSON</b>	<code>{"T":305,"name":"boot","step":2,"json":{"T":205,"freq":500,"duration":30}}</code>	Replace a command in a mission file
<b>CMD_DELETE_STEP</b>	<code>{"T":306,"name":"boot","step":2}</code>	Delete a specific command from mission file
<b>CMD_RUN_STEP</b>	<code>{"T":307,"name":"boot","step":2}</code>	Execute a specific step
<b>CMD_RUN_MISSION</b>	<code>{"T":308,"name":"boot","interval":1000,"loop":1}</code>	Run a mission file; loop=-1 for infinite repeat
<b>CMD_DELETE_MISSION</b>	<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.