Essae Weighing Scale & Label Printer - User Space Driver Documentation

Prepared By	Venkatesh M	venkatesh.muninagaraju@essae.com
Reviewed By	Nagarajan V	nagarajan@essae.com

1. System Overview

This document describes the architecture, command interface, communication protocol, and usage of the **user-space Linux driver** for the **Essae weighing scale** and **label printer**, designed for RK3568-based systems.

The system supports two primary functionalities:

- Reading weight from a serially connected Essae weighing scale
- Printing labels using an Essae label printer by parsing .LFT label template files and combining with JSON product data

The system operates in two modes:

- **Server Mode**: TCP server on port 8888 to handle remote commands from GUI or other tools.
- The server is also configured to run in the background as a **systemd service**, always listening on port 8888. Users only need to run the Python GUI to interact with it.

2. Ports & Device Paths

Component	Port Path	Baud Rate	Description
Label Printer	/dev/ttyS0	115200	Receives formatted print commands
Weighing Scale	/dev/ttyS4	9600	Sends/receives ASCII or binary weight commands
Server	TCP Port 8888	-	Accepts remote printer/scale requests from clients

3. Communication Protocol Server Mode

The server listens on TCP port 8888. Based on the command received, it performs either:

- **Printer Mode:**

MODE:PRINTER

<JSON file path>	**Ex: Select The .JSON File Path In Essae_WSLPR_client.py**
<LFT slot number>	**EX: Select Label Desine File slot (1-99) from SQL_LFT_Files.db**
<selected barcode ID (1-99)>	

- **Scale Mode:**

MODE:WEIGHT

<command>	**Ex: RD_WEIGHT, XC_TARE, XC_REZERO etc..**

Each command is followed by response(s) sent over the same socket.

4. Weighing Scale Commands

The following commands are parsed and sent over /dev/ttyS4:

Command	Description	ASCII/Hex Value
RD_WEIGHT	Reads current weight	0x05
XC_TARE	Sends 'T' command (Tare)	'T' / 't'
XC_REZERO	Sends 'Z' command (Zero scale)	0x10
XC_SON	Enter calibration mode	0x12
XC_KEYCALxxxxx	Set calibration weight in grams	0x13
XC_CALZERO	Calibration - Zero	0x14
XC_CALSPAN	Calibration - Span	0x15
XC_CALIBRATE	Finalize calibration	0x16
XC_RDRAWCT	Read raw count	0x11
XC_RESTART	Restarts the scale	0x1C
RD_TECHSPEC	Read technical parameters	0x19
WR_TECHSPEC	Write technical parameters	0x18
RD_CUSSPEC	Read custom configuration	0x1B
WR_CUSSPEC	Write custom configuration	0x1A

Each command is sent as ASCII bytes. Delays and responses are handled with timeouts and retries.

5. Label Printer Commands

The printer receives formatted .LFT files containing drawing/printing instructions Refer the **Label Report Description language R2 PDF**. Each command starts with a tilde ~.

Command	Description
~S	Label size (~S,width_mm,height_mm)
~A	Define clear area
~T	Print fixed text
~V	Print variable (from JSON) text
~B	Print barcode data (from JSON)
~R	Draw rectangle
~C	Draw circle
~c	Escape Codes
~d	Print bitmap image
~I	Set print intensity (100-140)
~Y	Delay printing for specified ms
~P	Print page (finalize buffer)
~e	Read response from printer
~s	Line Spacing

6. Command Format Details Refer Label Report Description language R2.PDF

~T - Fixed Text

Command, x location [mm], y location [mm], angle [0: 90:180:270], font [1:2], x magnify [1-6], y magnify [1-6], text, data length, offset, justify [N: L: C: R], lines, line spacing [mm], mode [C: F: W: X: I: U], Pint Status [0-5]

~T,11.875,0.125,0,2,1,2,Heritage Fresh,14,0,N,1,2.500,E,1

~V - Variable Text

Command, x location [mm], y location [mm], angle [0: 90: 180: 270], font [1: 2], x magnify [1-6], y magnify [1-6], data ID, data, data length, offset, justify [N: L: C: R],

lines, line spacing [mm], mode [C: F: W: X: I: U], Pint Status [0-5]

~V,0,5.25,0,2,1,1,2,Onion Medium,36,0,C,1,2.500,W,1

~B - Barcode from JSON

Command, x location [mm], y location [mm], angle [0, 90, 180, 270], HRI font [1, 2], barcode width [0.125, 0.250, 0.375, 0.500, 0.625], barcode height [mm], data,

data length, offset, justify [N: L: C: R], barcode type [EAN13, CODE128, CODE128A, CODE128B, CODE128C: QRCODE], HRI position [N: T: B: 2],

mode [C: F: W: X: I], Pint Status [0-5],

~B,1.25,7.5,0,2,0.25,8,1234567890123456789012345678,28,0,N,CODE128,B,W,1,

7. Data Flow

Printing:

Client GUI → TCP → Server → Parse LFT No + JSON → Open /dev/ttyS0 → Send ESC/POS Commands → Label Printer

Weighing:

Client GUI → TCP → Server → Send Weighing command over /dev/ttyS4 → Get scale response → Return to client

8. GUI Python Client Integration

The **Essae_WSLPR_client.py** GUI connects to the server via TCP on port 8888 and provides:

- **Label printer LFT slot management:**
  - Add LFT: Upload .LFT to SQLite DB (SQL_LFT_Files.db)
  - Edit LFT: Edit .LFT content inside GUI
  - Delete LFT: Remove a slot from SQL database .db
  - Rename LFT: Rename slot name in database (optional enhancement)
- Barcode selection: 1–99 JSON entry number
- Print function: Sends .LFT slot, JSON, and barcode No via TCP port 8888
- Weighing scale commands:
  - Read weight
  - Tare / Zero / Restart
  - Calibration (multi-step)
  - Technical and custom spec I/O

The GUI embeds styling, user-friendly controls, and live logs per action.

Other features:

- Shows response logs for each tab (Normal, Calibration, Tech)
- Displays current connection status
- Blocks print if JSON not selected or not connected
- Uses database file **SQL_LFT_Files.db** with **lft_files (slot, name, content)** schema

To run the GUI:

```
$ python3 Essae_WSLPR_client.py
```

9. Required Packages & Dependencies

Install the following packages on Ubuntu 22.04:

For Server:

```
$ sudo apt install build-essential libjson-c-dev libsqlite3-dev
```

For Client GUI:

```
$ sudo apt install python3 python3-pyqt5 sqlitebrowser
```

```
$ pip3 install PyQt5
```

10. Server, and Service Modes

Server Mode:

Run as TCP server and handle requests:

```
$ ./Essae_WSLPR_server
```

Service Mode:

To run the server persistently in the background on every boot, configure it as a systemd service.

Create the file:

```
$ sudo nano /etc/systemd/system/essae_server.service
```

Paste the following:

```
[Unit]
Description=Essae WSLPR TCP Server
After=network.target

[Service]
ExecStart=/home/essae/Documents/Essae_Data/Projects/Essae_Rockchip_RK3568_Seavo/Essae_WSLPR_Driver_Code/Essae_WSLPR_server
Restart=always
User=essae
WorkingDirectory=/home/essae/Documents/Essae_Data/Projects/Essae_Rockchip_RK3568_Seavo/Essae_WSLPR_Driver_Code

[Install]
WantedBy=multi-user.target
```

Reload and start:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable essae_server.service
$ sudo systemctl start essae_server.service
```

Check status:

```
$ journalctl -u essae_server.service --no-pager --since "5 minutes ago"
```

Ensure the .db path used by the service is correct. If needed, make the DB path absolute in code or place the .db file in the same working directory.

Then, just run the GUI client normally:

```
$ python3 Essae_WSLPR_client.py
```

11. Error Handling

- Missing serial ports are logged as warnings but don't crash the server
- Invalid .LFT commands are skipped
- JSON decoding uses libjson-c, with fallback defaults if keys are missing
- GUI disables print when disconnected, logs all failures

📎 Appendix

Database File: SQL_LFT_Files.db

- Table: lft_files(slot INTEGER PRIMARY KEY, name TEXT, content BLOB)
- Used to save .LFT data uploaded via GUI
- Accessible using sqlitebrowser or sqlite3 on Ubuntu