Esphone & Lesp

Manual

Process Flow and Language Reference

---

1 Overview

Esphone is a proof-of-concept project to demonstrate the feasibility of single-task CLI-based operating systems, running on ESP32 microcontroller with an ST7735 TFT as the main display and an SD card as the external storage. It features a custom Lisp-like interpreter called **Lesp** to evaluate scripts at runtime.

This manual provides required hardware specifications, detailed explanation of the project pipeline and scripting language documentation, while the corresponding software is available on GitHub.

2 Hardware Requirements[1]

- **ESP32 microcontroller**
- **ST7735 TFT Display** (on VSPI)
  - CS: GPIO 5
  - DC: GPIO 22
  - RST: GPIO 21
- **SD Card** (on HSPI)
  - CS: GPIO 27
  - MOSI: GPIO 13
  - MISO: GPIO 26
  - SCK: GPIO 14

[1]The following pin scheme is based on the source code. Users may use a different configuration as they wish.

3 System Architecture

3.1 Process Flow

The process is quite simple. At start, ESP32 initializes a Serial connection, SD card via HSPI, ST7735 via VSPI, creates a mutex (thread safety) and displays shell prompt. Then Esphone takes user input via serial input and when enter is hit, command prompt is split into two parts: filename and arguments. Following that, the system forwards the arguments, if any, to the main function located in *<filename>.txt* (similar to *main* → *main.exe* in Windows, for now the extension remains as txt, to be changed). Finally, the Lesp interpretator executes the main function with the given parameters.

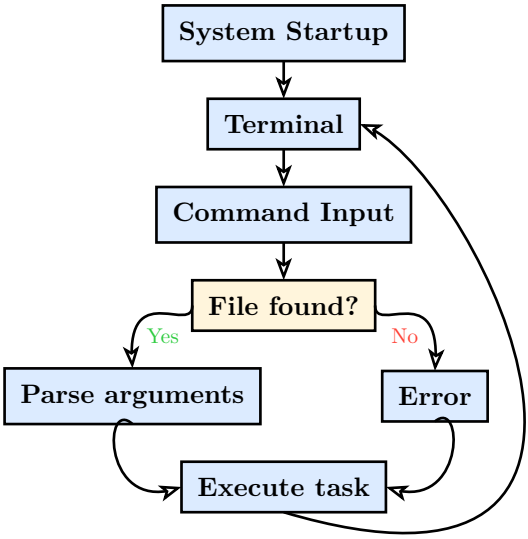

Figure 1: Esphone System Process Flow

4 Lesp reference

4.1 Data Types

Type	Description	Example
Integer	Whole numbers	`42`, `-10`
Float	Decimal numbers	`3.14`, `-0.5`
String	Text in double quotes	`"hello"`
Symbol	Identifiers/variable names	`x`, `my-var`
List	Ordered collection	`(1 2 3)`
Function	Built-in function	`+`, `print`
Lambda	User-defined function	`(lambda (x) (* x x))`
Nil	Empty/null value	`nil`

Table 1: Lesp Data Types

4.2 Syntax

4.2.1 Comments

Not explicitly supported in current implementation.

4.2.2 S-Expressions

All code is written as S-expressions, similar to Lisp (parenthesized lists):

```
(function arg1 arg2 arg3)
```

4.2.3 String Literals

```
"This is a string"
```

4.2.4 Numeric Literals

```
42        ; integer
3.14      ; float
-10       ; negative integer
```

5 Core Language Features

5.1 Basic operations

5.1.1 `def` - Define Variable

```
(def variable-name value)
```

Defines a new variable or lambda function in the current environment.

Example:

```
(def x 10)
(def name "Alice")
```

5.1.2 `set!` - Update Variable

```
(set! variable-name new-value)
```

Updates an existing variable in the environment chain.

Example:

```
(set! x 20)
```

5.1.3 `begin` - Sequential Execution

```
(begin expr1 expr2 expr3 ...)
```

Evaluates expressions one by one, returns the last result. Creates a new local scope. Is mainly used in loops or lambda functions for multiline bodies.

Example:

```
(begin
  (def x 5)
  (def y 10)
  (+ x y))  ; returns 15
```

5.1.4 `if` - Conditional Execution

```
(if condition then-expr else-expr)
```

Evaluates condition; if truthy (non-zero), evaluates then-expr, otherwise else-expr. Elif is currently not supported; however can be implemented using chained if statements.

Example:

```
(if (< x 10)
    (println "Small")
    (println "Large"))
```

5.1.5 `while` - Loop

```
(while condition body)
```

Repeatedly evaluates body while condition is truthy.

Example:

```
(def i 0)
(while (< i 5)
  (begin
    (println i)
    (set! i (+ i 1))))
```

5.1.6 `lambda` - Function Definition

```
(lambda (param1 param2 ...) body)
```

Creates an anonymous function.

Example:

```
(def square (lambda (x) (* x x)))
(square 5)  ; returns 25
```

5.1.7 `include` - Load Library

```
(include library-name)
```

Loads a built-in library or script from SD card. Libraries are loaded only once.

Built-in libraries: `math`, `sys`, `fs`, `wifi`, `http`

Example:

```
(include math)
(include wifi)
```

5.2 Library System

Libraries create namespaced environments. You can access library functions using dot notation:

```
(include math)
(math.sqrt 16)  ; returns 4.0
(math.sin math.pi)
```

6 Core functions

6.1 Arithmetic Operators

6.1.1 + - Addition

```
(+ num1 num2 ...)
```

Returns sum of all arguments. Result is float if any argument is float.

Example:

```
(+ 1 2 3)      ; returns 6
(+ 1.5 2)      ; returns 3.5
```

6.1.2 - - Subtraction

```
(- num1 num2 ...)
```

Subtracts subsequent arguments from the first.

Example:

```
(- 10 3 2)     ; returns 5
```

6.1.3 * - Multiplication

```
(* num1 num2 ...)
```

Returns product of all arguments.

Example:

```
(* 2 3 4)      ; returns 24
```

6.1.4 / - Division

```
(/ num1 num2 ...)
```

Divides first argument by subsequent arguments. Always returns float.

Example:

```
(/ 10 2)        ; returns 5.0
(/ 10 3)        ; returns 3.333...
```

6.2 Comparison Operators

6.2.1 < - Less Than

```
(< num1 num2)
```

Returns 1 if num1 $<$ num2, otherwise 0.

6.2.2 <= - Less Than or Equal

```
(<= num1 num2)
```

6.2.3 >= - Greater Than or Equal

```
(>= num1 num2)
```

6.2.4 = - Equality

```
(= val1 val2)
```

Compares two values for equality. Works with numbers, strings.

Example:

```
(= 5 5)             ; returns 1
(= "a" "a")         ; returns 1
(= 5 "5")           ; returns 0 (different types)
```

6.3 Logical Operators

6.3.1 `not` - Logical NOT

```
(not value)
```

Returns 1 if value is 0, otherwise returns 0.

6.3.2 `and` - Logical AND

```
(and val1 val2 ...)
```

Returns 1 if all values are truthy, otherwise 0.

6.3.3 `or` - Logical OR

```
(or val1 val2 ...)
```

Returns 1 if any value is truthy, otherwise 0.

6.4 Type Conversion

6.4.1 `int` - Convert to Integer

```
(int value)
```

Converts string to integer.

Example:

```
(int "42")     ; returns 42
```

6.4.2 `float` - Convert to Float

```
(float value)
```

Converts integer or string to float.

Example:

```
(float 42)       ; returns 42.0
(float "3.14")   ; returns 3.14
```

6.4.3 `string` - Convert to String

```
(string value)
```

Converts number to string representation.

6.4.4 `type` - Get Type Name

```
(type value)
```

Returns type as string: “int”, “float”, “string”, “list”, “function”, or “nil”.

6.5 List Operations

6.5.1 `list` - Create List

```
(list item1 item2 ...)
```

Creates a list from arguments.

Example:

```
(def mylist (list 1 2 3 4))
```

6.5.2 `get` - Get List Element

```
(get list index)
```

Returns element at index (0-based).

Example:

```
(get mylist 0)    ; returns first element
```

6.5.3 `set` - Set List Element

```
(set list index value)
```

Returns new list with element at index set to value.

Example:

```
(def newlist (set mylist 0 99))
```

6.5.4 `len` - Get List Length

```
(len list)
```

Returns number of elements in list.

6.5.5 `push` - Append to List

```
(push list value)
```

Returns new list with value appended.

6.5.6 `pop` - Remove Last Element

```
(pop list)
```

Returns new list with last element removed.

6.5.7 `slice` - Extract Sublist

```
(slice list start end)
```

Returns new list containing elements from start (inclusive) to end (exclusive).

Example:

```
(slice (list 1 2 3 4 5) 1 4)  ; returns (2 3 4)
```

6.6 String Operations

6.6.1 `strlen` - String Length

```
(strlen string)
```

6.6.2 `concat` - Concatenate Strings

```
(concat str1 str2 ...)
```

Combines all string arguments into one.

Example:

```
(concat "Hello" " " "World")  ; returns "Hello World"
```

6.6.3 `substr` - Substring

```
(substr string start length)
```

Extracts substring starting at index with given length.

6.6.4 `charAt` - Character at Index

```
(charAt string index)
```

Returns single-character string at index.

6.6.5 `split` - Split String

```
(split string delimiter)
```

Splits string by delimiter, returns list of substrings.

Example:

```
(split "a,b,c" ",")  ; returns ("a" "b" "c")
```

6.7 Input/Output

6.7.1 `print` - Print Without Newline

```
(print value1 value2 ...)
```

Prints values to TFT display.

6.7.2 `println` - Print With Newline

```
(println value1 value2 ...)
```

Prints values followed by newline.

Example:

```
(println "Hello, World!")
(println "x = " x)
```

7 Math Library

Load with: `(include math)`

7.1 Functions

7.1.1 `sqrt` - Square Root

```
(math.sqrt number)
```

7.1.2 `abs` - Absolute Value

```
(math.abs number)
```

7.1.3 `pow` - Power

```
(math.pow base exponent)
```

7.1.4 `min` - Minimum

```
(math.min num1 num2 ...)
```

7.1.5 `max` - Maximum

```
(math.max num1 num2 ...)
```

7.2 Trigonometric Functions

All angles in radians.

- `(math.sin angle)` - Sine
- `(math.cos angle)` - Cosine
- `(math.tan angle)` - Tangent
- `(math.arcsin value)` - Arcsine
- `(math.arccos value)` - Arccosine
- `(math.arctan value)` - Arctangent

7.3 Constants

- `math.pi` - π (3.14159…)
- `math.e` - Euler's number (2.71828…)

Example:

```
(include math)
(def radius 5)
(def area (* math.pi (math.pow radius 2)))
(println "Area: " area)
```

8 System Library

Load with: `(include sys)`

8.1 `sys.cls` - Clear Screen

```
(sys.cls)
```

Clears the TFT display.

8.2 `sys.time` - Get Milliseconds

```
(sys.time)
```

Returns milliseconds since startup.

Example:

```
(def start (sys.time))
; ... do work ...
(def elapsed (- (sys.time) start))
```

8.3 `sys.delay` - Delay Execution

```
(sys.delay milliseconds)
```

Pauses execution for specified milliseconds.

8.4 `sys.exit` - Exit Script

```
(sys.exit)
```

Terminates the current script task.

9 Filesystem Library

Load with: `(include fs)`

All paths are relative to SD card root. Do not include leading slash.

9.1 `fs.exists` - Check File Exists

```
(fs.exists filename)
```

Returns 1 if file exists, 0 otherwise.

Example:

```
(if (fs.exists "data.txt")
    (println "File found")
    (println "File not found"))
```

9.2 `fs.read` - Read File

```
(fs.read filename)
```

Returns file contents as string, or empty string on error.

Example:

```
(def content (fs.read "config.txt"))
(println content)
```

9.3 `fs.write` - Write File

```
(fs.write filename content)
```

Writes string to file (overwrites existing). Returns 1 on success, 0 on failure.

Example:

```
(fs.write "output.txt" "Hello, World!")
```

9.4 `fs.append` - Append to File

```
(fs.append filename content)
```

Appends string to end of file. Returns 1 on success.

9.5 `fs.remove` - Delete File

```
(fs.remove filename)
```

Deletes file. Returns 1 on success.

10 WiFi Library

Load with: `(include wifi)`

10.1 `wifi.connect` - Connect to Network

```
(wifi.connect ssid password)
```

Connects to WiFi network. Returns 1 on success, 0 on failure.

Example:

```
(include wifi)
(if (wifi.connect "MyNetwork" "password123")
    (println "Connected!")
    (println "Connection failed"))
```

10.2 `wifi.disconnect` - Disconnect

```
(wifi.disconnect)
```

10.3 `wifi.status` - Get Connection Status

```
(wifi.status)
```

Returns WiFi status code.

10.4 `wifi.ip` - Get IP Address

```
(wifi.ip)
```

Returns IP address as string, or empty string if not connected.

Example:

```
(println "IP: " (wifi.ip))
```

10.5 `wifi.scan` - Scan Networks

```
(wifi.scan)
```

Returns list of available SSID strings.

Example:

```
(def networks (wifi.scan))
(def i 0)
(while (< i (len networks))
  (begin
    (println (get networks i))
    (set! i (+ i 1))))
```

11 HTTP Library

Load with: `(include http)`

11.1 `http.get` - HTTP GET Request

```
(http.get url)
```

Performs HTTP/HTTPS GET request. Returns response body as string.

Example:

```
(include http)
(include wifi)

(wifi.connect "MyWiFi" "password")
(def response (http.get "https://api.example.com/data"))
(println response)
```

11.2 `http.get-file` - Download File

```
(http.get-file url local-path)
```

Downloads file from URL to SD card. Returns 1 on success, 0 on failure.

Example:

```
(http.get-file "https://example.com/data.json" "/data.json")
```

11.3 `http.status` - Get Last Status Code

```
(http.status)
```

Returns HTTP status code from last request.

11.4 `http.response` - Get Last Response

```
(http.response)
```

Returns response body from last request as string.

12 Complete Example Programs

12.1 Hello World

```
(def main (lambda ()
  (println "Hello, World!")))
```

12.2 Calculator with Arguments

```
; File: calc.txt
(def main (lambda (arg1 arg2)
      (begin
        (def a (int arg1))
        (def b (int arg2))
        (println "Sum: " (+ a b))
        (println "Product: " (* a b)))))
```

Run: `calc 5 10`

12.3 Temperature Converter

```
; File: temp.txt
(include math)

(def c-to-f (lambda (c)
  (+ (* c 1.8) 32)))

(def main (lambda (args)
      (begin
        (def celsius (float args))
        (def fahrenheit (c-to-f celsius))
        (println celsius "C = " fahrenheit "F"))))
```

Run: `temp 25`

12.4 Factorial Calculator

```
; File: factorial.txt
(include math)

(def factorial (lambda (n)
  (if (<= n 1)
      1
```

```
      (* n (factorial (- n 1))))))

(def main (lambda (args)
      (begin
        (def n (int args))
        (def result (factorial n))
        (println "Factorial of " n " is " result))))
```

Run: `factorial 5`

12.5 Circle Area Calculator

```
; File: circle.txt
(include math)

(def area (lambda (radius)
  (* math.pi (math.pow radius 2))))

(def main (lambda (args)
      (begin
        (def r (float args))
        (println "Area = " (area r)))))
```

Run: `circle 7.5`

13 Technical Details

13.1 Memory Management

- Uses C++ heap allocation for dynamic structures
- Lambdas and their closures persist in memory
- No automatic garbage collection - avoid creating excessive objects in loops
- Script task allocated 16KB stack on FreeRTOS Core 1

13.2 Threading

- Main loop runs on Core 0
- Each script executes in dedicated FreeRTOS task on Core 1
- Terminal output protected by mutex for thread safety
- Only one script runs at a time

13.3 Limitations

- No garbage collection

- Lists and strings are immutable (operations return new copies)
- No proper tail-call optimization
- Limited error handling/reporting
- No debugger or stack traces
- Variables cannot be truly deleted, only reassigned

13.4 Terminal Display

- 6x8 pixel characters
- Auto-wraps at screen edge
- Auto-scrolls when screen fills (clears entire screen)

13.5 File Format

- Scripts must be plain text files (`.esp` in the future)
- Stored on SD card with `.txt` extension
- Carriage returns (`\r`) automatically stripped
- UTF-8 encoding recommended

14 Conclusion

Esphone is created to show that implementing a single-task operating system-ish programs is doable on ESP32, and serves its purpose well. However, since now it is just a proof of concept, newer updates should aim to transform the idea into a fully functional standalone device by adding other critical standard libraries, optimizing the code, fixing system security (which currently doesn't exist; user can download and execute arbitrary code snippets) and writing a package manager.

Feel free to contribute!