Quick Reference

Tips & Troubleshooting

- **Communication UART (Rx, Tx)** can be configured in the System Settings.
- **Boot log and debug messages** are available on pin **GPIO2** (P2) at 115200 baud, 1 stop bit, no parity. Those messages may be disabled through compile flags.
- **Loopback test**: Connect the Rx and Tx pins with a piece of wire. Anything you type in the browser should appear on the screen. Set *Parser Timeout = 0* in Terminal Settings to be able to manually enter escape sequences.
- There is very little RAM available to the webserver, and it can support at most 4 connections at the same time. Each terminal session (open window with the terminal screen) uses one persistent connection for screen updates. **Avoid leaving unused windows open**, or either the RAM or connections may be exhausted.
- **For best performance**, use the module in Client mode (connected to external network) and minimize the number of simultaneous connections. Enabling AP consumes extra RAM because the DHCP server and Captive Portal DNS server are started.
- In AP mode, **check that the WiFi channel used is clear**; interference may cause flaky connection. A good mobile app to use for this is WiFi Analyzer (Google Play). Adjust the hotspot strength and range using the *Tx Power setting*.
- Hold the BOOT button (GPIO0 to GND) for ~1 second to force enable AP. Hold it for ~6 seconds to restore default settings. (This is indicated by the blue LED rapidly flashing). Default settings can be overwritten in the System Settings.

Basic Intro & Nomenclature

ESPTerm emulates VT102 (pictured) with some additions from later VT models and Xterm. All commonly used attributes and commands are supported. ESPTerm is capable of displaying ncurses applications such as *Midnight Commander* using *agetty*.

ESPTerm accepts UTF-8 characters received on the communication UART and displays them on the screen, interpreting some codes as Control Characters. Those are e.g. *Carriage Return* (13), *Line Feed* (10), *Tab* (9), *Backspace* (8) and *Bell* (7).

Escape sequences start with the control character *ESC* (27), followed by any number of ASCII characters forming the body of the command.

Nomenclature & Command Types

Examples on this help page use the following symbols for special characters and command types:

(spaces are for clarity only, *DO NOT* include them in the commands!)

Name	Symbol	ASCII	C string	Function
ESC	`\e`	`ESC` (27)	`"\e"`, `"\x1b"`, `"\033"`	Introduces an escape sequence. *(Note:* `\e` *is a GCC extension)*
Bell	`\a`	`BEL` (7)	`"\a"`, `"\x7"`, `"\07"`	Audible beep
String Terminator	`ST`	`ESC \` (27 92) *or* `\a` (7)	`"\x1b\\"`, `"\a"`	Terminates a string command (`\a` can be used as an alternative)
Control Sequence Introducer	`CSI`	`ESC [`	`"\x1b["`	Starts a CSI command. Examples: `\e[?7;10h`, `\e[2J`
Operating System Command	`OSC`	`ESC ]`	`"\x1b]"`	Starts an OSC command. Is followed by a command string terminated by `ST`. Example: `\e]0;My Screen Title\a`
Select Graphic Rendition	`SGR`	`CSI n;n;nm`	`"\x1b[1;2;3m"`	Set text attributes, like color or style. 0 to 10 numbers can be used, `\e[m` is treated as `\e[0m`

There are also some other commands that don't follow the CSI, SGR or OSC pattern, such as `\e7` or `\e#8`. A list of the most important escape sequences is presented in the following sections.

Screen Behavior & Refreshing

The initial screen size, title text and button labels can be configured in Terminal Settings.

Screen updates are sent to the browser through a WebSocket after some time of inactivity on the communication UART (called "Redraw Delay"). After an update is sent, at least a time of "Redraw Cooldown" must elapse before the next update can be sent. Those delays are used is to avoid burdening the server with tiny updates during a large screen repaint. If you experience issues (broken image due to dropped bytes), try adjusting those config options. It may also be useful to try different baud rates.

User Input: Keyboard, Mouse

Keyboard

The user can input text using their keyboard, or on Android, using the on-screen keyboard which is open using a button beneath the screen. Supported are all printable characters, as well as many control keys, such as arrows, *Ctrl+letters* and function keys. Sequences sent by function keys are based on VT102 and Xterm.

The codes sent by *Home*, *End*, *F1-F4* and cursor keys are affected by various keyboard modes (*Application Cursor Keys*, *Application Numpad Mode*, *SS3 Fn Keys Mode*). Some can be set in the Terminal Settings, others via commands.

Here are some examples of control key codes:

Key	Code	Key	Code
Up	`\e[A`, `\eOA`	F1	`\eOP`, `\e[11~`
Down	`\e[B`, `\eOB`	F2	`\eOQ`, `\e[12~`
Right	`\e[C`, `\eOC`	F3	`\eOR`, `\e[13~`
Left	`\e[D`, `\eOD`	F4	`\eOS`, `\e[14~`
Home	`\eOH`, `\e[H`, `\e[1~`	F5	`\e[15`
End	`\eOF`, `\e[F`, `\e[4~`	F6	`\e[17~`
Insert	`\e[2~`	F7	`\e[18~`
Delete	`\e[3~`	F8	`\e[19~`
Page Up	`\e[5~`	F9	`\e[20~`
Page Down	`\e[6~`	F10	`\e[21~`
Enter	`\r` (13)	F11	`\e[23~`
Ctrl+Enter	`\n` (10)	F12	`\e[24~`
Tab	`\t` (9)	ESC	`\e` (27)
Backspace	`\b` (8)	Ctrl+A..Z	ASCII 1-26

Action buttons

The blue buttons under the screen send ASCII codes 1, 2, 3, 4, 5, which incidentally correspond to *Ctrl+A,B,C,D,E*. This choice was made to make button press parsing as simple as possible.

Mouse

ESPTerm implements standard mouse tracking modes based on Xterm. Mouse tracking can be used to implement powerful user interactions such as on-screen buttons, draggable sliders or dials, menus etc. ESPTerm's mouse tracking was tested using VTTest and should be compatible with all terminal applications that request mouse tracking.

Mouse can be tracked in different ways; some are easier to parse, others more powerful. The coordinates can also be encoded in different ways. All mouse tracking options are set using option commands: `CSI ? n h` to enable, `CSI ? n l` to disable option *n*.

Mouse Tracking Modes

All tracking modes produce three numbers which are then encoded and send to the application. First is the *event number* N, then the *X and Y coordinates*, 1-based.

Mouse buttons are numbered: 1=left, 2=middle, 3=right. Wheel works as two buttons (4 and 5) which generate only press events (no release).

Option	Name	Description
`9`	**X10 mode**	This is the most basic tracking mode, in which **only button presses** are reported. N = button - 1: (0 left, 1 middle, 2 right, 3, 4 wheel).
`1000`	**Normal mode**	In Normal mode, both button presses and releases are reported. The lower two bits of N indicate the button pressed: `00b` (0) left, `01b` (1) middle, `10b` (2) right, `11b` (3) button release. Wheel buttons are reported as 0 and 1 with added 64 (e.g. 64 and 65). Normal mode also supports tracking of modifier keys, which are added to N as bit masks: 4=*Shift*, 8=*Meta/Alt*, 16=*Control/Cmd*. Example: middle button with *Shift* = 1 + 4 = `101b` (5).
`1002`	**Button-Event tracking**	This is similar to Normal mode (`1000`), but mouse motion with a button held is also reported. A motion event is generated when the mouse cursor moves between screen character cells. A motion event has the same N as a press event, but 32 is added. For example, drag-drop event with the middle button will produce N = 1 (press), 33 (dragging) and 3 (release).
`1003`	**Any-Event tracking**	This mode is almost identical to Button Event tracking (1002), but motion events are sent even when no mouse buttons are held. This could be used to draw on-screen mouse cursor, for example. Motion events with no buttons will use N = 32 + *11b* (35).
`1004`	**Focus tracking**	Focus tracking is a separate function from the other mouse tracking modes, therefore they can be enabled together. Focus tracking reports when the terminal window (in Xterm) gets or loses focus, or in ESPTerm's case, when any user is connected. This can be used to pause/resume a game or on-screen animations. Focus tracking mode sends `CSI I` when the terminal receives, and `CSI O` when it loses focus.

Mouse Report Encoding

The following encoding schemes can be used with any of the tracking modes (except Focus tracking, which is not affected).

Option	Name	Description

Option	Name	Description
--	**Normal encoding**	This is the default encoding scheme used when no other option is selected. In this mode, a mouse report has the format `CSI M n x y`, where *n*, *x* and *y* are characters with ASCII value = 32 (space) + the respective number, e.g. 0 becomes 32 (space), 1 becomes 33 (!). The reason for adding 32 is to avoid producing control characters. Example: `\e[M !!` - left button press at coordinates 1,1 when using X10 mode.
`1005`	**UTF-8 encoding**	This scheme should encode each of the numbers as a UTF-8 code point, expanding the maximum possible value. Since ESPTerm's screen size is limited and this has no practical benefit, this serves simply as an alias to the normal scheme.
`1006`	**SGR encoding**	In SGR encoding, the response looks like a SGR sequence with the three numbers as semicolon-separated ASCII values. In this case 32 is not added like in the Normal and UTF-8 schemes, because it would serve nor purpose here. Also, button release is not reported as 11b, but using the normal button code while changing the final SGR character: `M` for button press and `m` for button release. Example: `\e[2;80;24m` - the right button was released at row 80, column 24.
`1015`	**URXVT encoding**	This is similar to SGR encoding, but the final character is always `M` and the numbers are like in the Normal scheme, with 32 added. This scheme has no real advantage over the previous schemes and was added solely for completeness.

Alternate Character Sets

ESPTerm implements Alternate Character Sets as a way to print box drawing characters and special symbols. A character set can change what each received ASCII character is printed as on the screen (eg. "{" is "π" in codepage `0`). The implementation is based on the original VT devices.

Since ESPTerm also supports UTF-8, this feature is the most useful for applications which can't print UTF-8 or already use alternate character sets for historical reasons.

Supported codepages

- `B` - US ASCII (default)
- `A` - UK ASCII: # replaced with £
- `0` - Symbols and basic line drawing (standard DEC alternate character set)
- `1` - Symbols and advanced line drawing (based on DOS codepage 437, ESPTerm specific)

All codepages use codes 32-127, 32 being space. A character with no entry in the active codepage stays unchanged.

Codepage `A`

35 # £

Codepage `0`

96 ` ♦	97 a ▒	98 b ␉	99 c ␌	100 d ␍	101 e ␊	102 f °	103 g ±	104 h ␤	105 i ␋	106 j ┘	107 k ┐	108 l ┌	109 m └	110 n ┼
111 o ⎺	112 p ⎻	113 q ─	114 r ⎼	115 s ⎽	116 t ├	117 u ┤	118 v ┴	119 w ┬	120 x │	121 y ≤	122 z ≥	123 { π	124 | ≠	125 } £
126 ~ ·														

Codepage `1`

33 ! ☺	34 " ☻	35 # ♥	36 $ ♦	37 % ♣	38 & ♠	39 ' •	40 (⌛	41) ○	42 * ↯	43 + ♪	44 , ♫	45 - ☼	46 . ⌂	47 / ☢
48 0 ░	49 1 ▒	50 2 ▓	51 3 │	52 4 ┤	53 5 ╡	54 6 ╢	55 7 ╖	56 8 ╕	57 9 ╣	58 : ║	59 ; ╗	60 < ╝	61 = ╜	62 > ╛
63 ? ┐	64 @ └	65 A ┴	66 B ┬	67 C ├	68 D ─	69 E ┼	70 F ╞	71 G ╟	72 H ╚	73 I ╔	74 J ╩	75 K ╦	76 L ╠	77 M ═
78 N ╬	79 O ╧	80 P ╨	81 Q ╤	82 R ╥	83 S ╙	84 T ╘	85 U ╒	86 V ╓	87 W ╫	88 X ╪	89 Y ┘	90 Z ┌	91 [█	92 \ ▄
93] ▌	94 ^ ▐	95 _ ▀	96 ` ↕	97 a ↑	98 b ↓	99 c →	100 d ←	101 e ↔	102 f ▲	103 g ▼	104 h ►	105 i ◄	106 j ◢	107 k ◣
108 l ◤	109 m ◥	110 n ╭	111 o ╮	112 p ╯	113 q ╰	114 r r	115 s s	116 t t	117 u u	118 v v	119 w w	120 x x	121 y y	122 z z
123 { {	124 | |	125 } ✔	126 ~ ✘											

Switching commands

There are two character set slots, G0 and G1. Those slots are selected as active using ASCII codes Shift In and Shift Out (those originally served for shifting a red-black typewriter tape). Often only G0 is used for simplicity.

Each slot (G0 and G1) can have a different codepage assigned. G0 and G1 and the active slot number are saved and restored with the cursor and cleared with a screen reset (`\ec`).

The following commands are used:

Code	Meaning

Code	Meaning
`\e(`*x*	Set G0 = codepage *x*
`\e)`*x*	Set G1 = codepage *x*
SO (14)	Activate G0
SI (15)	Activate G1

Commands: Style SGR

All text attributes are set using SGR commands like `\e[10;20;30m`, with up to 10 numbers separated by semicolons. To restore all attributes to their default states, use SGR 0: `\e[0m` or `\e[m`.

Those are the supported text attributes SGR codes:

Style	Enable	Disable
Bold	1	21, 22
Faint	2	22
Italic	3	23
Underlined	4	24
Blink	5	25
Inverse	7	27
~~Striked~~	9	29
𝔉𝔯𝔞𝔨𝔱𝔲𝔯	20	23

Commands: Color SGR

Colors are set using SGR commands (like `\e[10;20;30m`). The following tables list the SGR codes to use. Selected colors are used for any new text entered, as well as for empty space when using line and screen clearing commands. The configured default colors can be restored using SGR 39 for foreground and SGR 49 for background.

The actual color representation depends on a color theme which can be selected in Terminal Settings.

Foreground colors

30	31	32	33	34	35	36	37
90	91	92	93	94	95	96	97

Background colors

40	41	42	43	44	45	46	47
100	101	102	103	104	105	106	107

Commands: Cursor Functions

The coordinates are 1-based, origin is top left. The cursor can move within the entire screen, or in the active Scrolling Region if Origin Mode is enabled.

After writing a character, the cursor advances to the right. If it has reached the end of the row, it stays on the same line, but writing the next character makes it jump to the start of the next line first, scrolling up if needed. If Auto-wrap mode is disabled, the cursor never wraps or scrolls the screen.

Legend: Italic letters such as *n* are ASCII numbers that serve as arguments, separated with a semicolon. If an argument is left out, it's treated as 0 or 1, depending on what makes sense for the command.

Movement

Code	Meaning

`\e[`*n*`A`
`\e[`*n*`B`
`\e[`*n*`C`
`\e[`*n*`D`
	Move cursor up (`A`), down (`B`), right (`C`), left (`D`)

`\e[`*n*`F`
`\e[`*n*`E`
	Go *n* lines up (`F`) or down (`E`), start of line

`\e[`*r*`d`
`\e[`*c*`G`
`\e[`*r*`;`*c*`H`
	Go to absolute position - row (`d`), column (`G`), or both (`H`). Use `\e[H` to go to 1,1.

`\e[6n`	Query cursor position. Sent back as `\e[`*r*`;`*c*`R`.

Save / restore

Code	Meaning

`\e[s`
`\e[u`
	Save (`s`) or restore (`u`) cursor position

`\e7` `\e8`	Save (`7`) or restore (`8`) cursor position and attributes

Scrolling Region

Code	Meaning
`\e[a;br`	Set scrolling region to rows *a* through *b* and go to 1,1. By default, the scrolling region spans the entire screen height. The cursor can leave the region using absolute position commands, unless Origin Mode (see below) is active.

`\e[?6h`
`\e[?6l`
	Enable (`h`) or disable (`l`) Origin Mode and go to 1,1. In Origin Mode, all coordinates are relative to the Scrolling Region and the cursor can't leave the region.

`\e[nS`
`\e[nT`
	Move contents of the Scrolling Region up (`S`) or down (`T`), pad with empty lines of the current background color. This is similar to what happens when AutoWrap is enabled and some text is printed at the very end of the screen.

Tab stops

Code	Meaning
`\eH`	Set tab stop at the current column. There are, by default, tabs every 8 columns.

`\e[nI`
`\e[nZ`
	Advance (`I`) or go back (`Z`) *n* tab stops or end/start of line. ASCII *TAB* (9) is equivalent to `\e[1I`

`\e[0g`
`\e[3g`
	Clear tab stop at the current column (`0`), or all columns (`3`).

Other options

Code	Meaning

`\e[?7h`
`\e[?7l`
	Enable (`h`) or disable (`l`) cursor auto-wrap and screen auto-scroll

`\e[?25h`
`\e[?25l`
	Show (`h`) or hide (`l`) the cursor

Commands: Screen Functions

Legend: Italic letters such as *n* are ASCII numbers that serve as arguments, separated with a semicolon. If an argument is left out, it's treated as 0 or 1, depending on what makes sense for the command.

Code	Meaning
`\e[mJ`	Clear part of screen. *m*: 0 - from cursor, 1 - to cursor, 2 - all
`\e[mK`	Erase part of line. *m*: 0 - from cursor, 1 - to cursor, 2 - all
`\e[nX`	Erase *n* characters in line.

Code	Meaning

`\e[`*n*`L`
`\e[`*n*`M`
	Insert (`L`) or delete (`M`) *n* lines. Following lines are pulled up or pushed down.

`\e[`*n*`@`
`\e[`*n*`P`
	Insert (`@`) or delete (`P`) *n* characters. The rest of the line is pulled left or pushed right. Characters going past the end of line are lost.

Commands: System Functions

It's possible to dynamically change the screen title text and action button labels. Setting an empty label to a button makes it look disabled. The buttons send ASCII 1-5 when clicked. Those changes are not retained after restart.

Code	Meaning
`\ec`	Clear screen, reset attributes and cursor. The screen size, title and button labels remain unchanged.
`\e[5n`	Query device status, ESPTerm replies with `\e[0n` "device is OK". Can be used to check if the terminal has booted up and is ready to receive commands.
CAN (24)	This ASCII code is not a command, but is sent by ESPTerm when it becomes ready to receive commands. When this code is received on the UART, it means ESPTerm has restarted and is ready. Use this to detect spontaneous restarts which require a full screen repaint.
`\e]0;`*t*`\a`	Set screen title to *t* (this is a standard OSC command)
`\e]`*80+n*`;`*t*`\a`	Set label for button *n* = 1-5 (code 81-85) to *t* - e.g.`\e]81;Yes\a` sets the first button text to "Yes".
`\e]`*90+n*`;`*m*`\a`	Set message for button *n* = 1-5 (code 81-85) to *m* - e.g.`\e]94;iv\a` sets the 3rd button to send string "iv" when pressed.

`\e[?800h`
`\e[?800l`
	Show (`h`) or hide (`l`) action buttons (the blue buttons under the screen).

`\e[?801h`
`\e[?801l`
	Show (`h`) or hide (`l`) menu/help links under the screen.

`\e[12h`
`\e[12l`
	Enable (`h`) or disable (`l`) Send-Receive Mode (SRM). SRM is the opposite of Local Echo, meaning `\e[12h` disables and `\e[12l` enables Local Echo.

`\e[8;`*r*`;`*c*`t`	Set screen size to *r* rows and *c* columns (this is a command borrowed from Xterm)