Interfacing the ESP8266 Wireless Terminal

Ondřej Hruška
Katedra měření, FEL ČVUT

March 2, 2017

Contents

1 Introduction

The purpose of this document is to present the ESP8266 Wireless Terminal firmware and describe how the module can be interfaced by an external microcontroller.

This document is divided into three sections: the first part explains the internal makeup of the module and it's possibilities, then we move on to the supported control sequences and details of the communication protocol, and in the last part the wireless settings are discussed.

2 Feature overview

The module implements a simple, VT100-compatible terminal emulator with a screen of up to 25x80 characters, controlled by ANSI escape sequences for colors, cursor movement and screen manipulation. It's capable of displaying received characters, as well as receiving input from the keyboard or mouse and sending those back over the serial line.

The user can access the terminal screen using their web browser thanks to a tiny built-in webserver, after connecting to the module over WiFi. Apart from the terminal display, there is also a help page and a configuration page for setting up the WiFi connection.

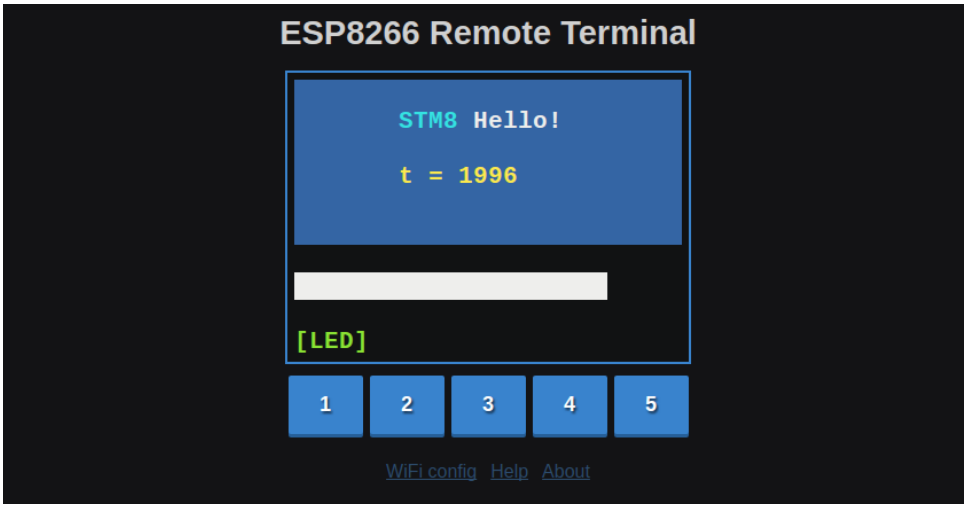

Figure 1: The terminal screen, displayed in a web browser

2.1 Terminal implementation

A simplified block diagram of the firmware is shown in Figure 2.

The terminal keeps in its memory the screen buffer, the cursor position, and the currently active colors and attributes. The screen size can be changed arbitrarily, with the limit of total 2000 cells, which corresponds to the resolution of 25x80. Each cell contains a single character and its foreground and background colors.

When a byte is received over the serial interface, it enters a state machine that interprets ANSI escape sequences and passes through any other characters. Then,

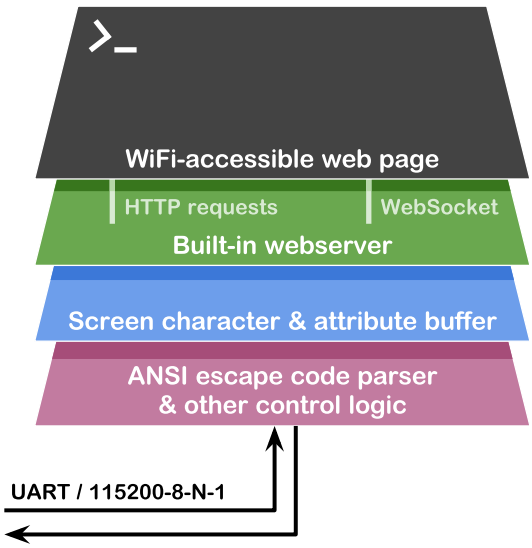

Figure 2: A high level block diagram of the terminal module

depending on the character, it's either discarded, treated as a control code (such as `'\r'` or `'\n'`), or written to the active cell.

After writing a character to the screen, the cursor is moved accordingly, wrapping to the next line if needed. Upon reaching the bottom right end of the screen, all the screen content is scrolled upwards to make room for the next line. This can be switched off, if needed, but is very useful for things like a scrolling log viewer. It also mirrors the behavior of desktop terminal emulators, albeit with no scrollback—the topmost line is lost forever.

Escape sequences, composed of the ASCII code 27 and one or more characters following it, can set text color, move the cursor, clear part of the screen etc. This will be discussed later in the document.

At the other end, facing the outside world, is a tiny webserver that renders the screen to a HTML page for whomever tries to access it. After the initial page load, the browser connects to the server via a WebSocket for real-time updates of the screen content. Any subsequent changes should be visible in the browser almost instantly, if the connection is good.

3 Interfacing the terminal

3.1 UART connection

Communication between the master controller and the WiFi module is performed by sending ASCII characters over a two-wire UART interface, running at 115200 baud with 8 data bits, 1 stop-bit and no parity.

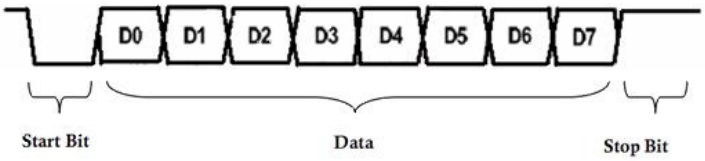

Figure 3: UART frame diagram (source)

Apart from a few custom messages (e.g. screen resize, mouse click), the implemented escape sequences are standard and should work in any proper terminal emulator. This means the module can be swapped for a USB-serial adapter and the user interface should still work without any changes (using appropriate PC software like PuTTY or GtkTerm). This also allows the user to develop their master controller firmware independently and add the WiFi module afterwards.

3.2 Debug port

The WiFi module has two UART ports. One is used for communication, the other is tx-only and is used by the firmware to print debug messages.

During development, or to get some insight into the inner workings, a USB-serial adapter can be connected to the output pin (marked DBG or GPIO2) to monitor those messages. Alternatively, a LED connected to the pin can be used as a crude activity indicator.

3.3 Control codes and escape sequences

As was said earlier, the communication consists of sending and receiving ASCII characters; for reference, see the attached ASCII table in Figure 4.

Some characters outside the readable range (32–126) have special meaning as control codes. This includes *CR* (carriage return, `'\r'`, ASCII 13) and *LF* (line feed, `'\n'`, ASCII 10). Those codes change the cursor position: return to the beginning of the line, and move one line down, respectively. Use them together as *CR LF* (`"\r\n"`) to print a new line. A special case is the *ESC* character (ASCII 27), which starts an *escape sequence*.

The terminal also supports backspace (ASCII 8), which moves the cursor one step back and replaces that character with blank (space, code 32). Tab (ASCII 9) moves the cursor to the closest higher multiple of 4, which may be useful for value printing.

3.3.1 Escape sequences

All interesting commands, such as changing colors, clearing screen or moving the cursor are sent as ANSI escape sequences, which are groups of characters started by the *ESC* character with ASCII code 27.

Dec	Hex	Char	Code
0	00		NUL
1	01	Btn	SOH
2	02		STX
3	03		ETX
4	04		EOT
5	05		ENQ
6	06		ACK
7	07	ST	BEL
8	08	Bksp	BS
9	09		HT
10	0A	\n	LF
11	0B		VT
12	0C		FF
13	0D	\r	CR
14	0E		SO
15	0F		SI
16	10		DLE
17	11		DC1
18	12		DC2
19	13		DC3
20	14		DC4
21	15		NAK
22	16		SYN
23	17		ETB
24	18		CAN
25	19		EM
26	1A		SUB
27	1B		ESC
28	1C		FS
29	1D		GS
30	1E		RS
31	1F		US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Figure 4: ASCII table, adapted from MSDN. Highlighted are the supported or used control characters.

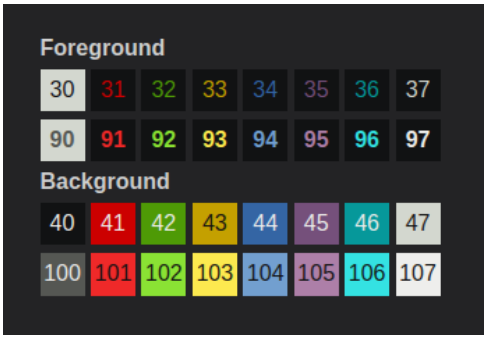

Figure 5: Screen color palette

For a comprehensive list of those sequences, please check the Wikipedia article on the topic. Almost all the listed commands are implemented, and some more. An up-to-date list of supported commands can be found on the built-in help page (there's a link under the terminal screen). Some details will be given in the following sections.

In this guide, the ESC character will be written as `'\e'`. It's a common, but non-standard notation used in C literals. Some compilers need `'\033'` or `'\x1b'` instead.

3.3.2 Colors and attributes

Text attributes are set using SGR commands (meaning *Set Graphic Rendition*). They all follow the format `\e[<codes>m`, where `<codes>` are up to 3 decimal numbers separated by semicolons (`;`). The code `\e[0m` resets all attributes to their default values.

There are, in total, **16 colors** available for both foreground and background. A color reference chart is pictured in Figure 5. The default color is gray on black, or `\e[37;40m`.

The terminal also supports the **negative** attribute (code 7), which swaps the colors when printing a character. This can be switched off by the **positive** attribute (code 27), or the reset code 0.

The **bold** attribute (code 1) makes the foreground color brighter, if used with the dimmer 30–37 colors; this is just for compatibility, there's no benefit over using the 90–37 and 100–107 color codes. The bright colors are always shown as bold in the web interface, for better visibility.

3.3.3 Cursor movement

The position where the next character will be written is given by the cursor coordinates. The screen coordinates are **1-based**, with origin $[1; 1]$ in the **top left** corner.

The cursor can be **hidden** using the standard sequence `\e[?25l` (ending with lowercase L). To show it again, use `\e[?25h`.

There are 8 cursor movement commands (A–H), the most useful of them being `\e[<y>;<x>H`, which sets the absolute position. The other commands are for **relative movement** (up `\e[<n>A`, down `\e[<n>B`, forward `\e[<n>C`, backward `\e[<n>D`) and n lines up or down with carriage return (down `\e[<n>E`, up `\e[<n>F`). The parameter `<n>` can be left out to default to 1.

3.3.4 Clearing commands

All the standard ANSI screen clearing commands are supported. When a part of the screen is cleared, the cells are set to space (ASCII 32) of the **currently selected colors**. That can be used for filling the screen with a color.

The `\e[<mode>J` command operates on the screen, `\e[<mode>K` on the current line.

`<mode>` can be 0 ("to the end"), 1 ("to the beginning"), or 2 ("clear all"). As an example, to clear the entire screen, use `\e[2J`.

To **reset** the screen to the default colors and move cursor to the top-left origin position, use `\ec` (no `[`, that's correct).

3.3.5 Screen scrolling

The screen content can be scrolled to make space for more text. This happens automatically when writing past the last line, but can also be done manually using the scrolling commands.

To scroll up, send `\e[<n>S`; to scroll down, send `\e[<n>T`. The parameter `<n>` is the number of lines, and can be left out to scroll just once.

To completely turn off line wrapping and automatic scrolling, use `\e[?7lm` (wrap OFF) and `\e[?7hm` (wrap ON);

3.3.6 Cursor memory

There is a memory slot available for saving and restoring the cursor position and attributes. To store and restore the cursor position *only*, use `\e[s` and `\e[u`. To store and restore the position *and attributes*, use `\e7` and `\e8`.

3.4 System commands

There is a set of commands that do not affect the screen directly, but are handled by the terminal module itself. They are again sent as escape sequences, to avoid confusing regular terminals used in place of the WiFi module.

Some of those are standard, some custom.

3.4.1 Query commands

The command `\e[6n` requests the **current cursor position**. It's sent back as `\e[<y>;<x>R`, parsing which is left as an exercise to the user.

To check if the device is ready, send `\e[5n`, and it should respond with `\e[0n`. This can be used for polling on startup, to check if the terminal is initialized. Conveniently, those are standard codes and work exactly the same way in desktop terminals like GtkTerm.

This is related to the magic code 0x18 ("CAN", 24) that the terminal sends to the UART on startup. The master controller can listen for this and re-init the module after restart automatically.

3.4.2 Changing screen size

The screen is backed by a static array of 2000 characters, which matches the standard DOS screen size of 80x25. You can set the size to anything within this range, eg. 40x40.

To set the size, send `\e]W<rows>;<cols>\a`, where `\a` is the BELL code (ASCII 7). Alternatively, instead of BELL, the ST sequence can be used: `\e\` (ESC+backslash).

Note the `]` in the above command - that's not a typo. The closing bracket marks "Operating System Command" in the ANSI standard, which can be used for custom commands like this.

3.4.3 Factory reset

If something goes wrong and the WiFi access can't be restored (i.e. by holding the BOOT pin at GND for 5 seconds to reset to AP mode), the persistent settings can be wiped by sending `\e]FR\a`. This resets the AP name and channel, forgets any saved SSID and password, and switches to AP mode.

Since this command is expected to be used manually, the terminal responds with `"FACTORY RESET\r\n"` and then restarts itself.

3.5 User input

Keyboard input is directly supported, sending keys typed at the terminal page transparently through the UART. Some keys generate sequences: arrows send escape sequences for relative cursor movement, enter sends `CR LF`, etc. *Keyboard input, naturally, does not work on mobile phones.*

Pressing the **numbered buttons** under the screen sends ASCII codes 1 to 5, chosen to be easy to process for the master controller.

There is also a proprietary **mouse support**. When a character on the screen is clicked or tapped, the terminal sends `\e[<y>;<x>M` to the master controller, with the row and column coordinates as decimal numbers, eg. `\e[7;12M`.

With some parsing and coordinate checking, this enables the implementation of things like virtual on-screen buttons for more advanced user interfaces. How to make use of this input is all in the hands of the master controller.

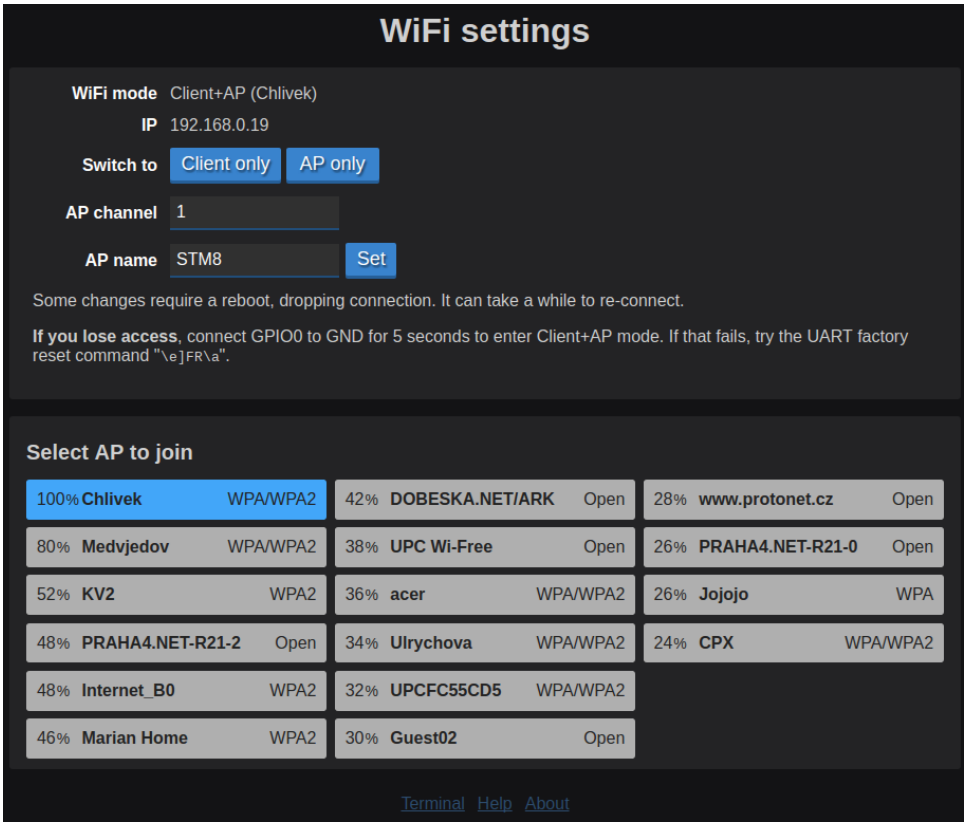

Figure 6: WiFi Settings page

4 WiFi configuration

WiFi can be configured using the built-in WiFi Settings page. The device can connect to an external network (Client mode), create it's own in access point (AP mode), or even both at once.

The internal AP is meant mostly for WiFi configuration; it's not the most stable and can become unreliable in noisy environments; on top of that, some mobile phones refuse to connect to it as there's (obviously) no internet connection.

It's therefore recommended to connect the module to an existing WiFi network and access it from within the network using it's received IP address. This address, obtained using DHCP, is printed to the debug log on startup, and is also shown on the WiFi setup page.

If the AP is disabled, it can be forced on by holding the BOOT (GPIO0) pin at GND for 5 seconds, then releasing it. The module will restart into the Client+AP mode, restoring access to the WiFi setup.

To access the settings page (shown in Figure 6), click at the header on the terminal screen, or use the navigation links at the bottom.

5 Useful links

- The Wireless Terminal Git repository & bugtracker
  https://github.com/MightyPork/esp-vt100-firmware
- Wikipedia page about ANSI escape sequences
  https://en.wikipedia.org/wiki/ANSI_escape_code
- VT-100 manual with a list of escape sequences
  http://vt100.net/docs/vt100-ug/chapter3.html