Programming hints for the ESP8266 Wireless Terminal

Ondřej Hruška
Katedra měření, FEL ČVUT

March 2, 2017

Contents

1 Introduction

The purpose of this document is to give some useful tips on writing the master controller firmware for interfacing the ESP8266 Wireless Terminal module.

Those are not meant as complete code examples, but rather to give some general guidance and push the reader in the right direction.

2 Waiting for the terminal to boot up

Before any control commands can be sent, it's necessary to wait for the WiFi module to initialize and start the terminal tasks.

This can be done using a fixed delay, but a more robust approach is to repeatedly send `\e[5n`, until `\e[0n` comes back. This is the "Device Status Query" command, and the response is "Device OK".

After the OK response arrives, it's safe to start sending text and control codes.

This works also with PC terminal emulators! It's a standard procedure, something like "ping".

2.1 A note on persistence

The ESP8266 firmware stores the WiFi settings in it's Flash memory, but anything else (screen size, background fill color etc) stay only in RAM. The module can sometimes restart itself (such as when there's a heap overflow, watchdog reset, changing WiFi mode etc.), and when that happens, the screen is reset to the default size and state.

Keep this in mind when building interactive user interfaces that do not repaint fully. A restart of the WiFi module can cause the screen content to appear corrupt.

To deal with this issue, the WiFi module is configured to send the ASCII code 24 ("CAN", 0x18) on startup. This lets the master controller listen for this special code and when it's received, re-init the screen size, background etc.

3 Parsing received characters

The UART input can easily be parsed with a Ragel-generated or hand-made state machine.

There are three kinds of messages to look out for:

1. Regular characters, 32–126
2. Control characters: 1–31, in particular CR, LF, BS and codes 1–5 for the blue buttons
3. Escape sequences, started with the ESC key.

   This includes arrow keys and the mouse click events (see the *Interfacing...* document for details).

   If you choose to implement those, it might be beneficial to check the *terminal's* source code (look for the `*.rl` Ragel file). There are going to be differences, in particular you only need a subset of the commands, but there will be some definite similarities.

   You may of course write your own parser, if you don't wish to deal with Ragel. As an example of a hand-written parser, check the example STM8 project source code. It's messy, all written in about an hour, but could get some ideas from it.

4 Useful links

- The Wireless Terminal Git repository & bugtracker
  `https://github.com/MightyPork/esp-vt100-firmware`
- The example STM8 project Git repository
  `https://github.com/MightyPork/esp-vt100-stm8-demo`
- Ragel, a state machine building tool
  `https://www.colm.net/open-source/ragel/`