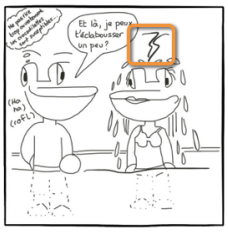
PicoFind - Level 1 - Writeup

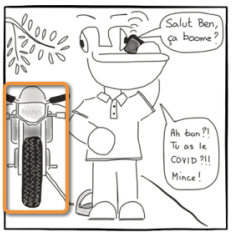
Obviously, you have to press **Start** in order to start. Then on each image, you have to find where the active zone is:



It is the same for the second comic:

And the third one:

For the last comic, things are a little bit more complicated. It is longer and there are two active zones on one of the images.

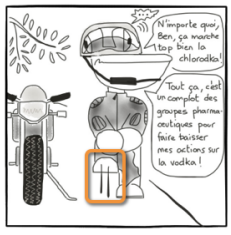


BD4 1/17

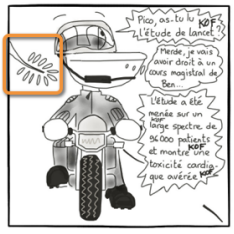
BD4 2/17

BD4 3/17

BD4 4/17



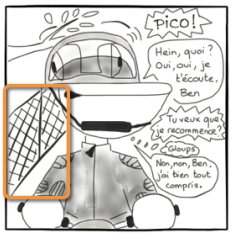
BD4 5/17



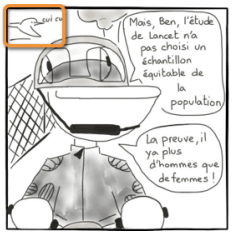
BD4 6/17

BD4 7/17

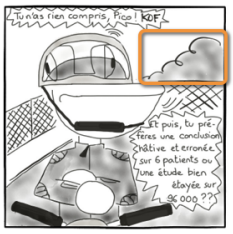
BD4 8/17



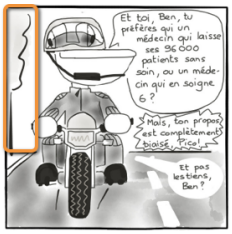
BD4 9/17



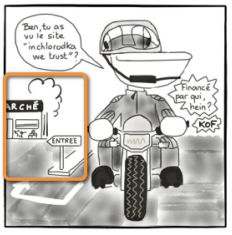
BD4 10/17



BD4 11/17

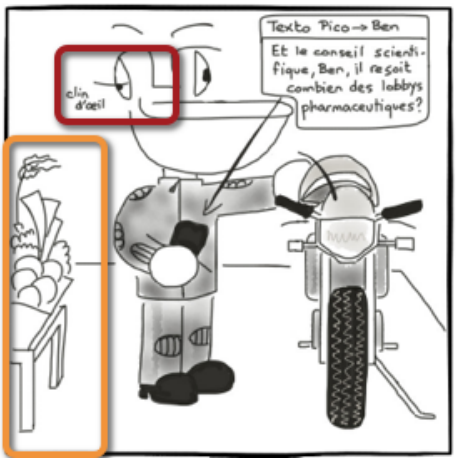


BD4 12/17



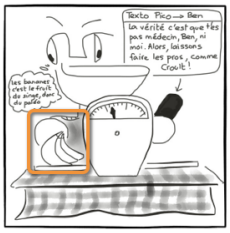
BD4 13/17

Here is the image with two zones: one in orange that continues to display the comic, and one in red.

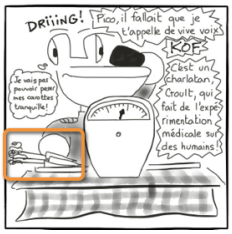


BD4 14/17

If you press on the wrong area, the next image of the comic is displayed:



BD4 15/17



BD4 16/17

BD4 17/17

At this step, you already go too far and you have to restart:

But if instead you have found the red zone, it displays the 1st flag:

PicoFind - Level 2 - Writeup

For the level 2, there is no way to display the flag by browsing the comics. You have to find another way. The screen has a serial to USB converter soldered and this is what you have to use.

Connecting a USB cable confirms that something is connected, but there is no prompt, no data sent even when you press on the screen. There are some hints in the description of the challenge:

> Author: Sebastien Andrivet DMT48270M043_05WT or DMT48270M043_02WT from DWIN Technologies

A Google search with `DMT48270M043_05WT` gives some datasheets, such as:

http://whiteelectronics.pl/Presta/index.php?controller=attachment&id_attachment=6

We can see that the serial port has a typical speed of 115200 bps and the interface is described as `8N1, 3.3V TTL/CMOS`.

Other Google requests give the following article:

http://sebastien.andrivet.com/en/posts/dwin-mini-dgus-display-development-guide-non-official/

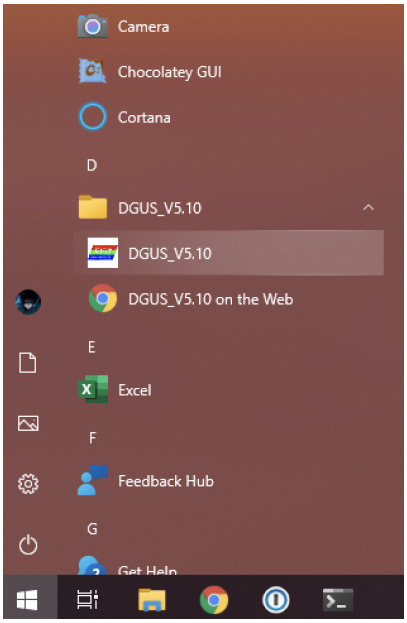
Note: This is not yet the case, I will make my best in the coming weeks to make this article indexed.

This document describes in detail how the LCD screen is working and mentions also an SDK that is downloadable from the article. There are mainly two solutions to solve this challenge:

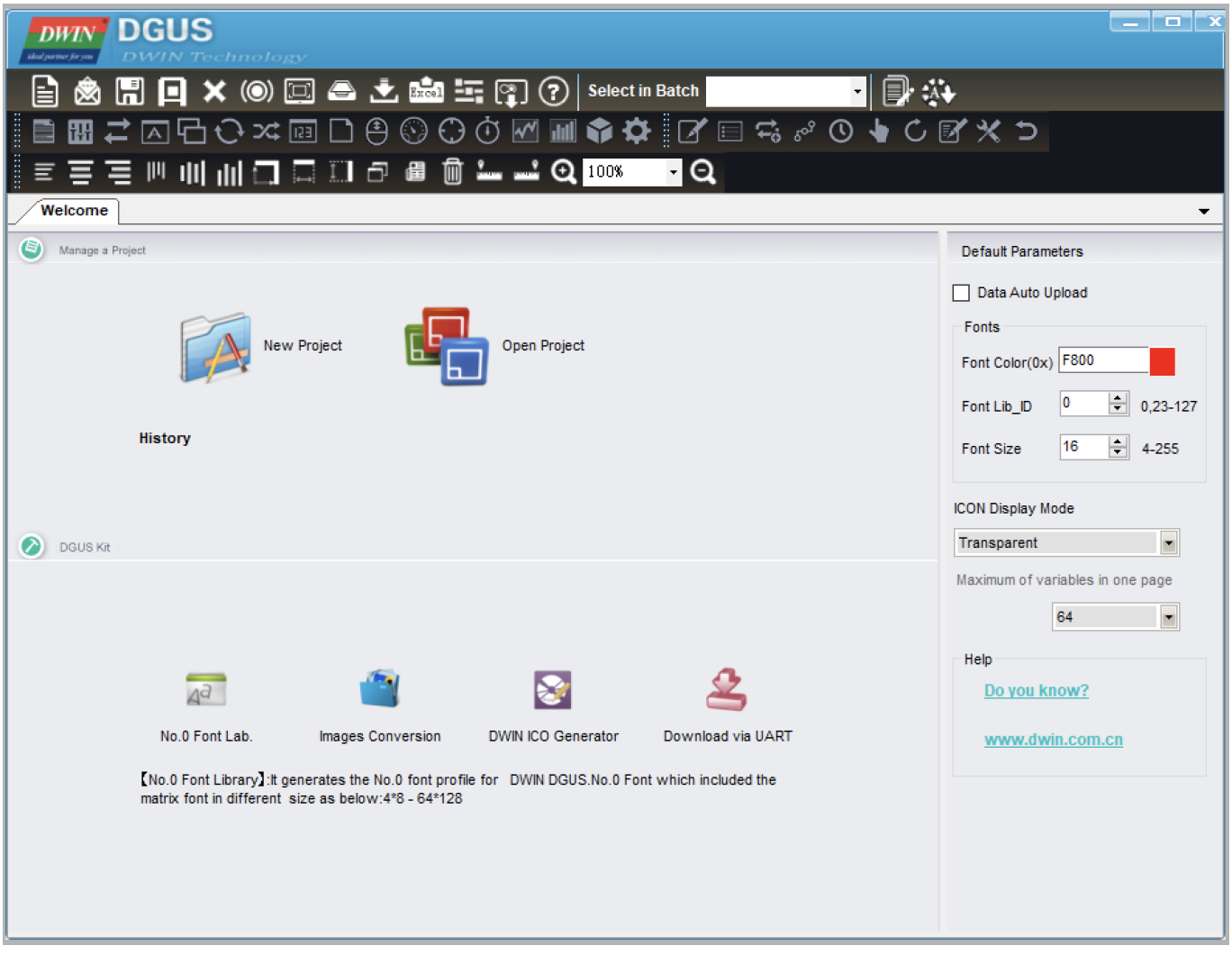
- The quick method by using one of the tools of the SDK
- The slow but more interesting method by using some programming

Solution 1 - Quick

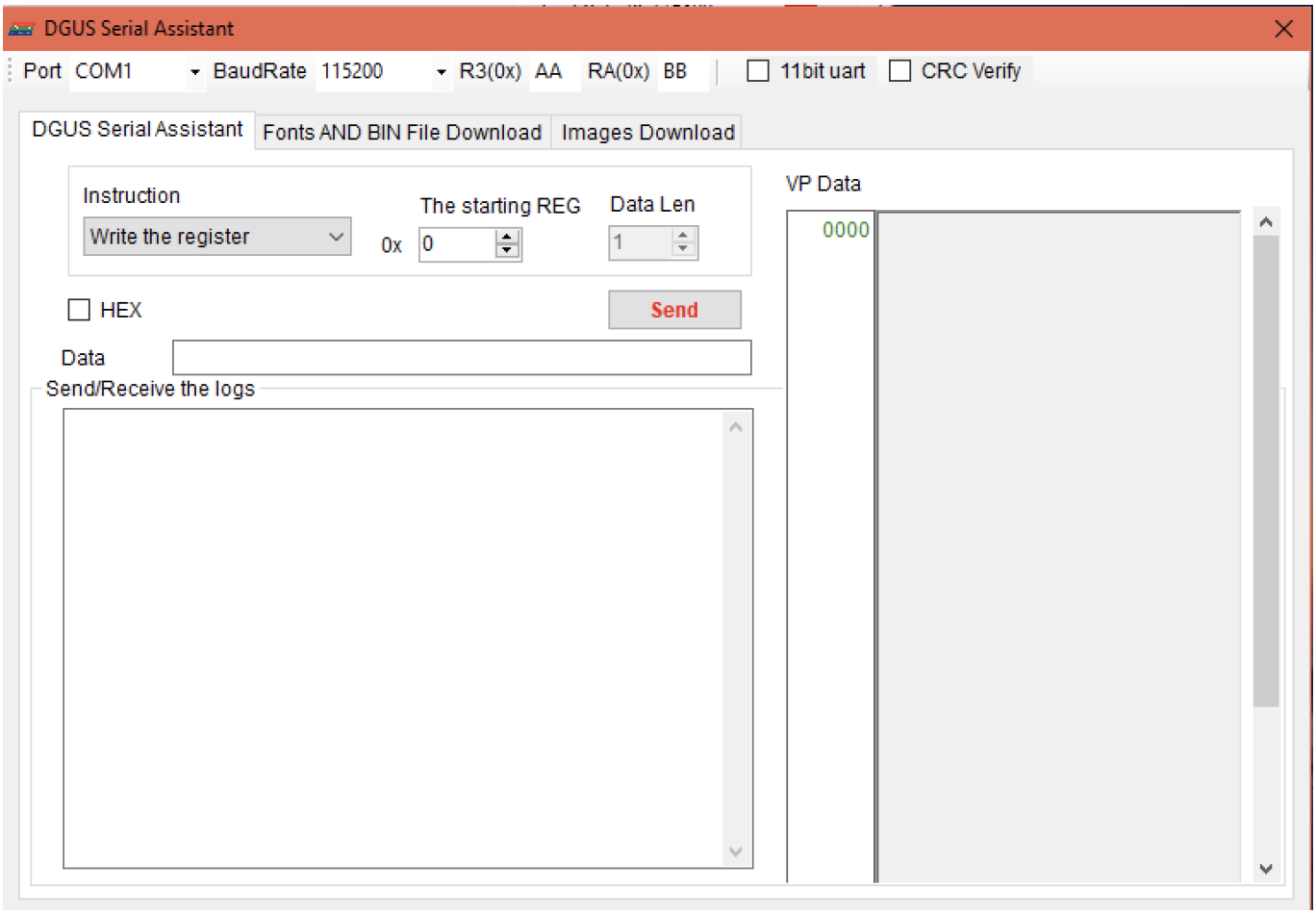
If you install the SDK (Windows only), you get the following application in the start menu:



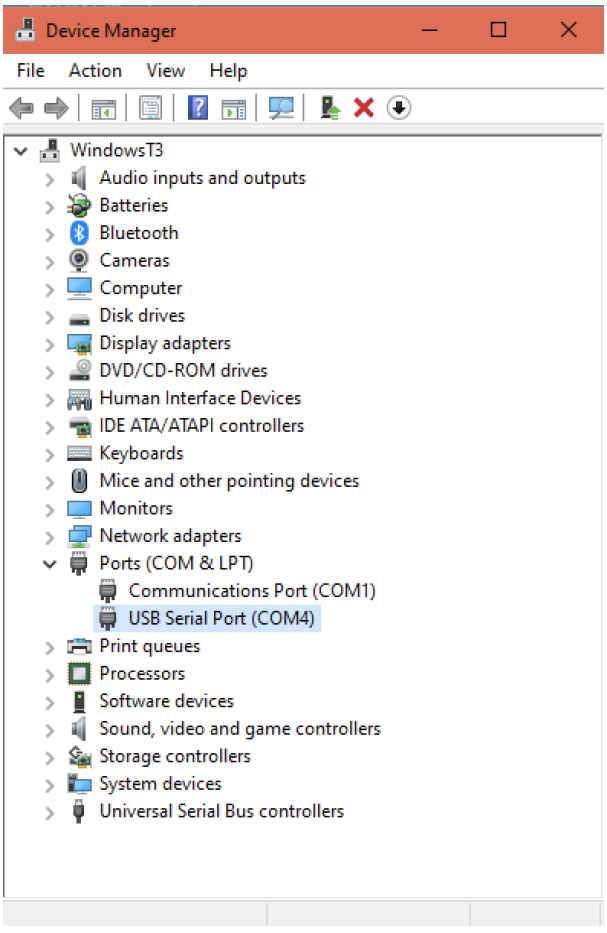
If you start this application, you get the following window:



The interesting item is "Download via UART". If you click on the icon, a new application starts:



So, this is apparently a tool to communicate with the LCD panel. We now have to find how. First, we can determine the port name by looking in Windows Device Manager:



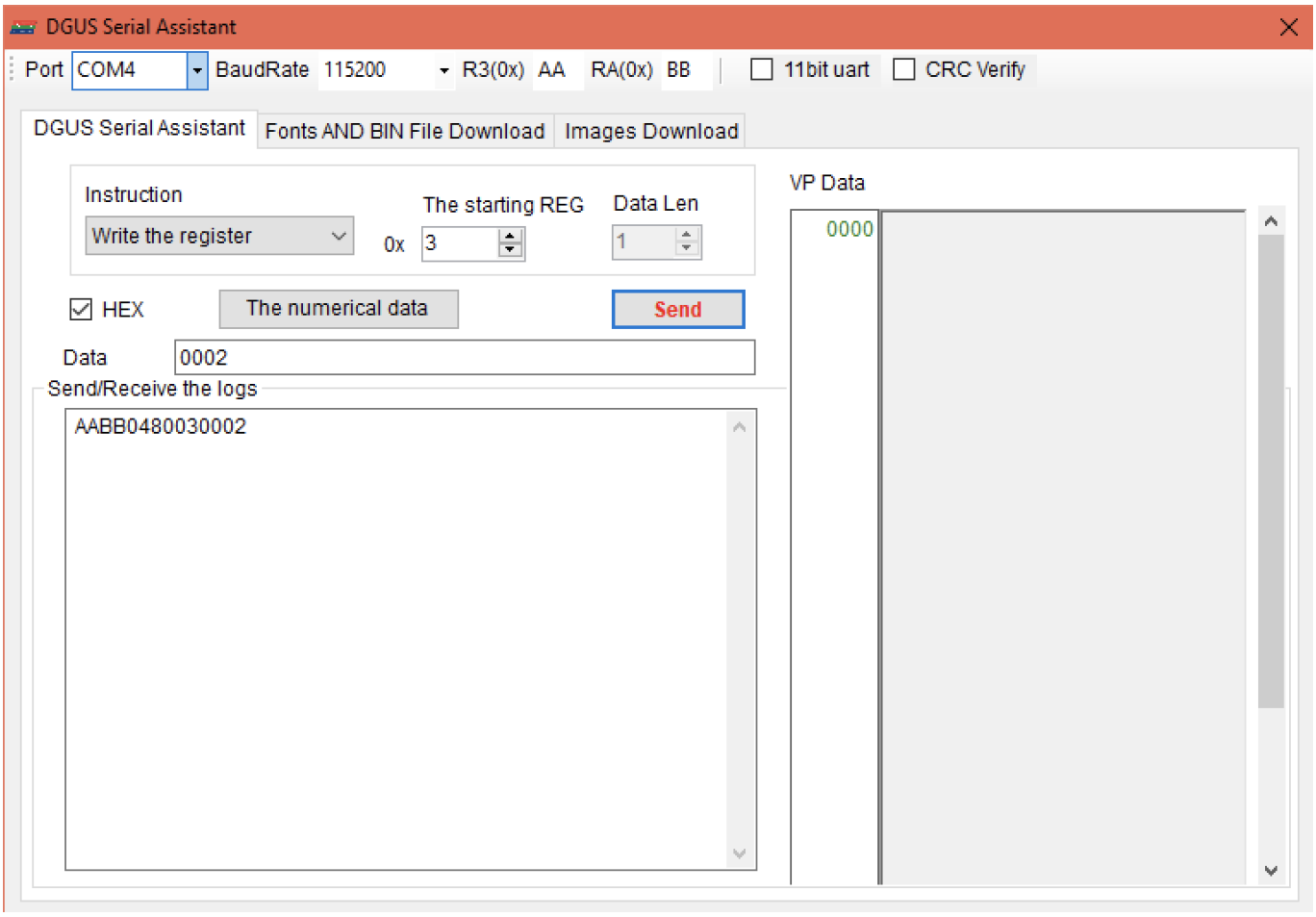
It is COM4 in this case. We now have to study the documentation of the panel. One relevant part is the paragraph 3.3 called "Registers":

3.3. Registers

The MINI DGUS Display provides 256 registers used to control the behavior of the system:

Address	Name	Size	Description
0x00	Version	1	MINI DGUS Display version number in BCD, 0x10 indicates 1.0
0x01	LED_NOW	1	LED brightness 0x00-0x40
0x02	BZ_TIME	1	Buzzer beeping time, x 10ms
0x03	PIC_ID	2	Read: Current picture ID, Write: Jump to picture

The register `PIC_ID` is at address 03 and is 2 bytes long. Let's try this in the communication tool. We select `COM4` for the port. Under **Instruction**, **Write the register** is already selected. For **Starting REG** we enter `0x03`, the address of `PIC_ID`. Then we enter a value for **Data** in hexadecimal (we check the box **HEX**). Which value? We don't know so let's try with `0002`:



Nothing changes on the LCD panel. We are doing something wrong. Let's go back to the documentation. The paragraph 4.2 "Data Frames" explains how to send frames to the panel:

4.2. Data Frames

Data frames consist of up to 4 blocks as described in this table:

Block	Name	Size	Comment
1	Header	2	Defined by `R3` & `RA`
2	Data length	1	Data length, including Command and Data (n+1)
3	Command	1	0x80-0x84
4	Data	n	Payload of the frame

Notes: *The default values for* `R3` *(*`UART_SYNC_H`*) and* `RA` *(*`UART_SYNC_L`*) are* `5A` *and* `A5`*. So the frames begin with the bytes* `5A A5`. The maximum data length that can be transmitted is 248 bytes. * It is not clear if MINI DGUS Displays are able to support CRC like DGUS displays. Some parts of the documentation suggest it is the case, but the data sheets suggest the contrary. I have assumed that MINI DGUS displays do not support CRC.

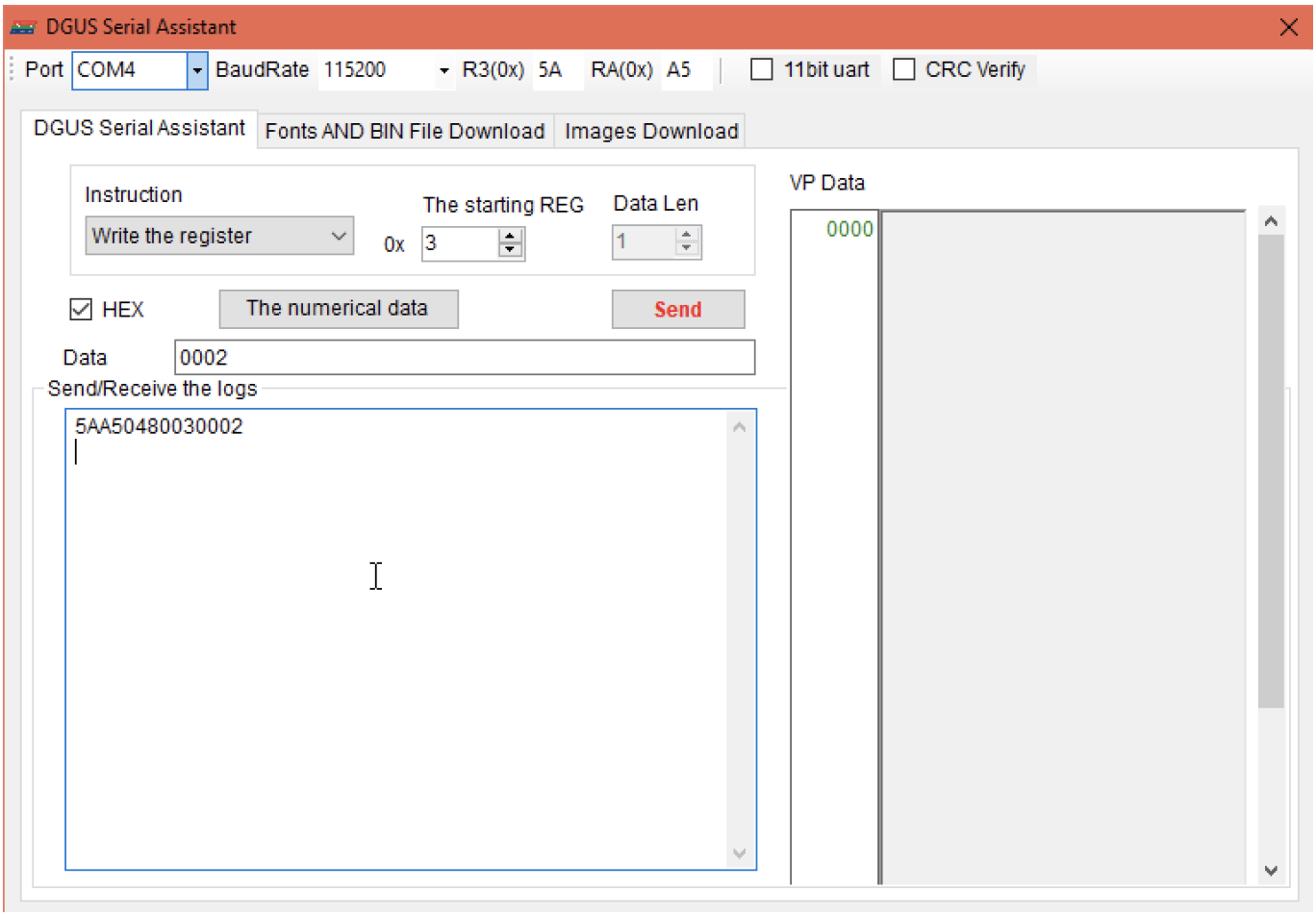
4.3 Commands

Command	Value	Data	Description
Write Register	0x80	REG + DATA	Write data into registers
Read Register	0x81	REG + LEN	Read data from registers

Response	Value	Data	Description
Read Register	0x81	REG + LEN + DATA	Response from the MINI DGUS Display

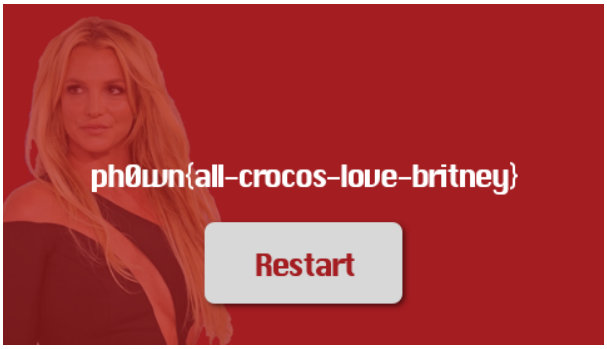
`REG` is between 0x00 and 0xFF and designates the first register to read of write. `LEN` is between 0x00 and 0xFF and designates the length (in **bytes**) of the data to read or write. `DATA` is the sequence of bytes to write or returned by the MINI DGUS Display.

We see that we have to send a header of two bytes which are determined by R3 and RA. It is also written that the values are 5A and A5 by default. In the communication tool, we have two inputs with R3 and RA but the values are AA and BB. So let's change those values:



And this time, it works! The page on the LCD panel changes:

If we try with other values, we see the other images and it is easy to find that the page with the flag is the image number `0x0021`:



Solution 2 - Slower but more interesting

Instead of installing the SDK and using a pre-made tool, we can program a small script to communicate with the LCD panel. Again, from the documentation, we see that we need to send a **Write Register** command with register **0x03** and the index of the image on two bytes. The format of the frame is the following:

4.2. Data Frames

Data frames consist of up to 4 blocks as described in this table:

Block	Name	Size	Comment
1	Header	2	Defined by `R3` & `RA`
2	Data length	1	Data length, including Command and Data (n+1)
3	Command	1	0x80-0x84
4	Data	n	Payload of the frame

Notes: *The default values for* `R3` (`UART_SYNC_H`) *and* `RA` (`UART_SYNC_L`) *are* `5A` *and* `A5`. *So the frames begin with the bytes* `5A A5`. The maximum data length that can be transmitted is 248 bytes. * It is not clear if MINI DGUS Displays are able to support CRC like DGUS displays. Some parts of the documentation suggest it is the case, but the data sheets suggest the contrary. I have assumed that MINI DGUS displays do not support CRC.

4.3 Commands

Command	Value	Data	Description
Write Register	0x80	REG + DATA	Write data into registers
Read Register	0x81	REG + LEN	Read data from registers

Response	Value	Data	Description
Read Register	0x81	REG + LEN + DATA	Response from the MINI DGUS Display

`REG` is between 0x00 and 0xFF and designates the first register to read of write. `LEN` is between 0x00 and 0xFF and designates the length (in **bytes**) of the data to read or write. `DATA` is the sequence of bytes to write or returned by the MINI DGUS Display.

So, we need to send the following sequence of bytes:

Offset	Name	Size	Value	Comment
0	Header	2	5A A5	R3 & RA
2	Data length	1	??	Data length
3	Command	1	80	Write Register
4	Data - REG	1	03	Register
5	Data - DATA	2	00 02	Image ID

This gives the bytes (in Big Endian as explained in the documentation):

```
5A A5 04 80 03 00 02
```

We can write a small program to send those bytes and increment the value of the image index. For example, in python using PySerial:

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import serial

BAUDRATE = 115200
COM_PORT = '/dev/tty.usbserial-A50285BI'  # macOS
# COM_PORT = '/dev/ttyS1'  # Linux
# COM_PORT = 'COM4'  # Windows


if __name__ == '__main__':
    with serial.Serial(COM_PORT, BAUDRATE) as ser:
        index = 0
        while True:
            input(f"Press Enter to display image # {index}...")
            frame = b'\x5A\xA5\x04\x80\x03\x00%c' % index
            ser.write(frame)
            index += 1
```