MMCommander - Developer's info

This document is a short explanation on how to use the MMCommander.

The MMCommander will describe itself as a USB CDC serial port and that's why if you're using Windows it should appear in the Device List as COMX, and as ttyUSBX or ttyACMX in Linux. When opening the serial port configure it as 57600 8N1.

In order to test the interface I would recommend to give it a try with RealTerm (in Windows) or moserial (in Linux).

Protocol description

The protocol I've designed is pretty simple:

If you send a **`0x00`** byte you will receive the FW version in one byte format.

To transmit:

- The first byte will be a transmission command. Depending on what you want to transmit you'll have to choose among these 3 options:
    - **`0x01`**: Transmit only the Payload.
    - **`0x81`**: Transmit the Payload and append its CRC-8 at the end.
    - **`0xC1`**: Transmit the Payload and append its CRC-16 at the end.
- The second byte is the number of bytes that you will send as Payload.
- The third byte is the number of times the MMCommander will repeat the transmission. This will be really helpful when sending "Turn on RF" commands to a pump, or to emulate a glucometer.
- Next will follow the Payload bytes.

When the MMCommander finishes the transmission process you will receive the 3 first bytes of your command as confirmation. In case the TxFilter is enabled and you try to transmit a "forbidden" payload or you have the NoTx version, you'll receive **`0x00`** as the number of bytes and the number of times, meaning that it wasn't transmitted.

When the MMCommander receives something it will report:

- First byte meaning the start of a reception:
    - **0x02** : Start of reception and a valid CRC-8 or CRC-16 was detected at the end.
    - **0x82** : Start of reception and no valid CRC-8 or CRC-16 was detected at the end.
- The second byte is the payload length in bytes
- After that comes the payload.

Tx Filter command: **0x03**

This command is only one byte long and is used to re-enable the transmission filter remotely. Whenever you disable or enable the transmission filter the MMCommander will send you one byte indicating the change in the state of the filter. When the filter is disabled you will receive **0x13** and, if it's re-enabled, you will receive **0x03**.

Repeat command: **0x04**

You'll receive a **0x04** byte each time the MMCommander receives another copy of the previous message. This way we can safe memory and bandwidth.

Simple, huh?

Simple transmission example

The best way to show you how it works is with an example:

First of all, you'll need a TxEnabled version for this to work. Let's say that you want to send the hexadecimal string **0xAB 0xCD 0xEF 0x99**. The first byte you should write to the serial port should be **0x01**, meaning that you want to start a transmission. The second byte will tell the MMCommander the number of bytes that you're passing as payload: in our case **0x04** bytes. The third byte represent the number of times that you want the MMCommander to repeat the transmission. As we only want to transmit it once: **0x01**.

What will you send to the MMCommander?

0x01 0x04 0x01 0xAB 0xCD 0xEF 0x99

After a while you have to receive **0x01 0x04 0x01** meaning that it was transmitted.

If you had a second MMCommander listening you should receive:

0x82 0x04 0xAB 0xCD 0xEF 0x99.

The first **0x82** means that something was received and no valid CRC was detected. The second byte **0x04** means that the payload is 4 bytes long and then the payload.

Why didn't I have a valid CRC?. Because I transmitted it without the CRC and the last bytes did not match the expected CRCs.

To send the string with a valid CRC-8 or CRC-16 (for Medtronic devices) we could send the same string changing the first byte to ask the MMCommander to calculate and append it.

Let's send: **0x81 0x04 0x01 0xAB 0xCD 0xEF 0x99**.

It seems the same, but the second MMCommander would receive:

0x02 0x05 0xAB 0xCD 0xEF 0x99 0x9B.

... adding **0x9B** to the payload as a valid CRC-8.

If we send: **0xC1 0x04 0x01 0xAB 0xCD 0xEF 0x99**

... the second will receive: **0x02 0x06 0xAB 0xCD 0xEF 0x99 0x06 0x13**.

Adding **0x06 0x13** as a valid CRC-16.

How to test that everything works fine

Well... I suppose that you only have one MMCommander so, what I would do, is make my pump transmit something. I've found that in my Paradigm Veo there's an option in *"Utilities -> Connect Devices -> Other Devices -> Find Device"* that will transmit what it seems to be like a discovery message each second (more or less).

You should receive messages with a length of **`0x0B`** bytes and a payload beginning with **`0xA2`**. If the message is received correctly, the CRC-8 should match and the first byte should be **`0x02`**.

Is there a sniffer?

If you're working in Linux, the answer is yes.

In the “src” folder you'll find a piece of SW called MMsimulator. Compile and run it passing the MMCommander device as the only parameter. It will open the device and log all the info received by the MMCommander.

It will also allow you to simulate other devices but... **PLEASE, DON'T USE IT IF YOU DON'T KNOW WHAT YOU'RE DOING. CAN BE DANGEROUS IF YOU USE REAL IDS**.

REMEMBER, USE IT AT YOUR OWN RISK