# NEO-6M Howto: change serial data rate via serial interface (boost from 9.6k to 115k) — for NEO-6, NEO-7, NEO-8

TLDR: it’s possible using internal command, even if docs say not

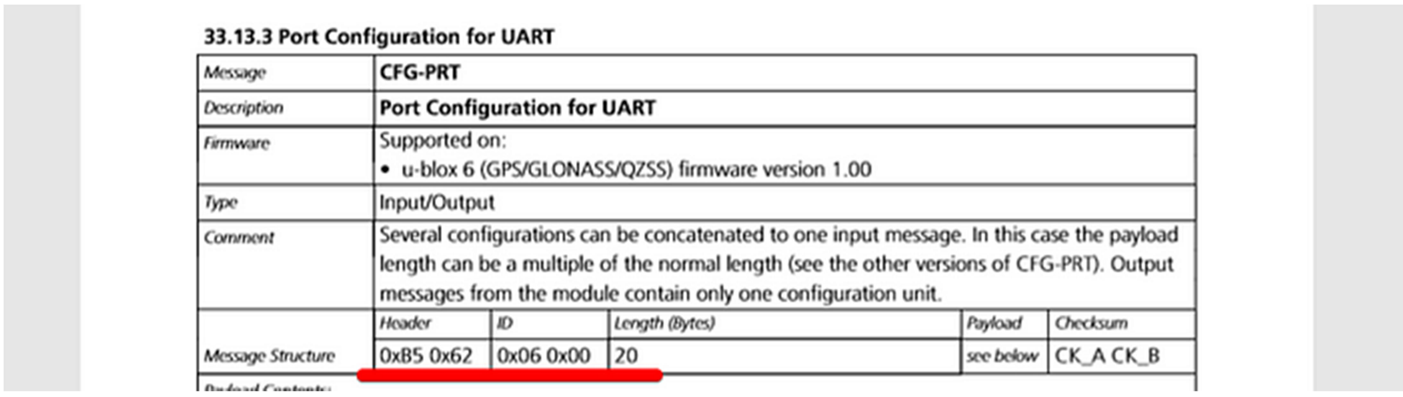
As you already noticed, we are able to tweak NEO GPS receiver to increase rate of measurements, from 1sec to 0.1sec, which gives us 10x drop in latency, but also 10x times of data must be sent using default serial protocol from device. This might be a problem, because some NMEA message packs (GPGSV) are quite large themselves — example shown at the end of the first article.

TTGO T-Beam ESP32 with NEO-6M onboard

**However, NEO supports much faster data rate on it’s interfaces,** just the serial default is set to = 9600, because it looks like “fits all solution”.

But the problem is, main doc spec states that serial data can’t be changed via serial itself, once it was started. *Sounds like nonsense for you? Me too.*

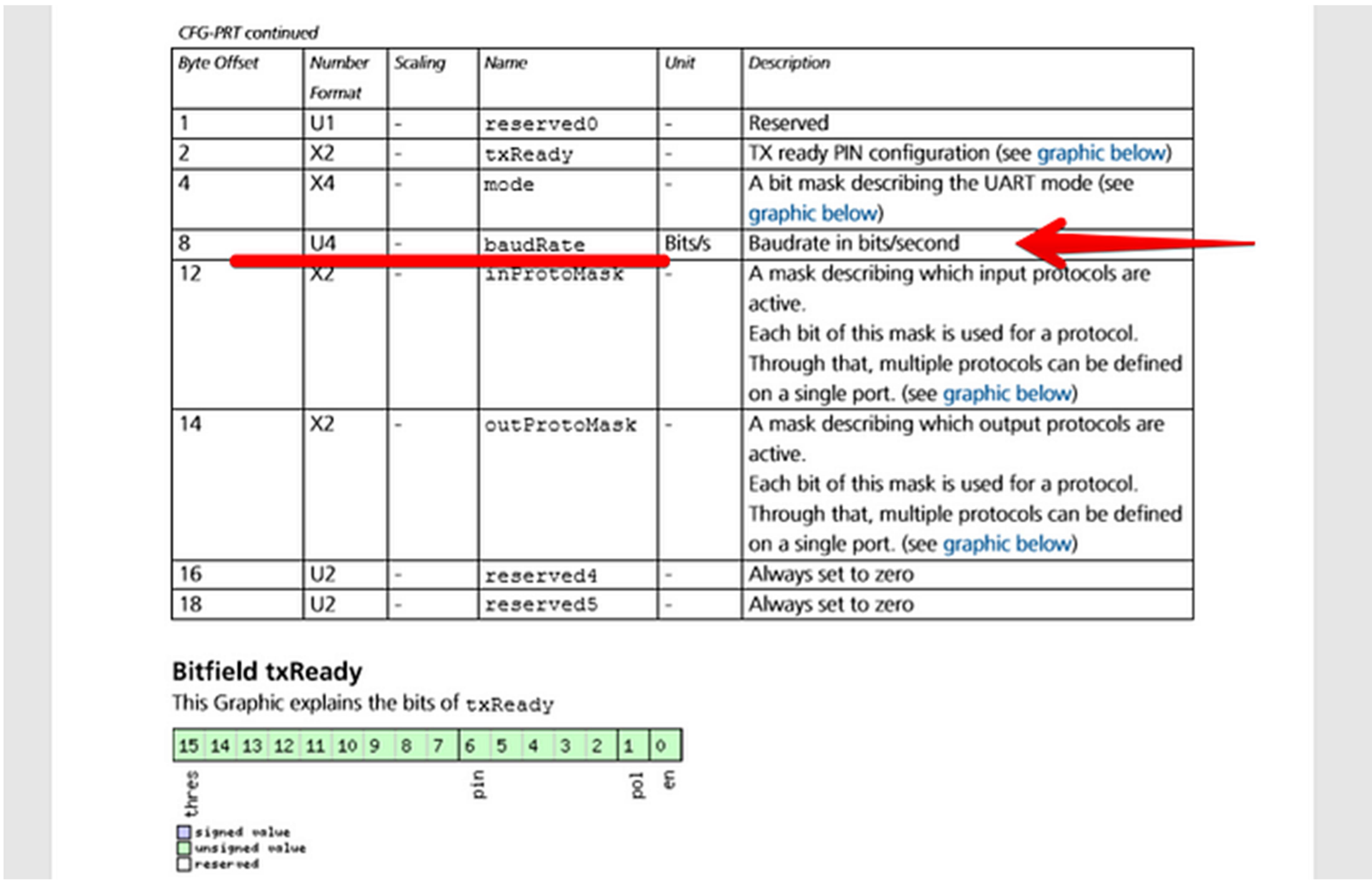
PUBX CFG-PRT message looks like this:



**33.13.3 Port Configuration for UART**

| | | | | | |
|---|---|---|---|---|---|
| *Message* | **CFG-PRT** | | | | |
| *Description* | **Port Configuration for UART** | | | | |
| *Firmware* | Supported on:<br>• u-blox 6 (GPS/GLONASS/QZSS) firmware version 1.00 | | | | |
| *Type* | Input/Output | | | | |
| *Comment* | Several configurations can be concatenated to one input message. In this case the payload length can be a multiple of the normal length (see the other versions of CFG-PRT). Output messages from the module contain only one configuration unit. | | | | |
| *Message Structure* | *Header* | *ID* | *Length (Bytes)* | *Payload* | *Checksum* |
| | 0xB5 0x62 | 0x06 0x00 | 20 | see below | CK_A CK_B |
| *Payload Contents:* | | | | | |

And here comes the baud rate:



*CFG-PRT continued*

| *Byte Offset* | *Number Format* | *Scaling* | *Name* | *Unit* | *Description* |
|---|---|---|---|---|---|
| 1 | U1 | - | `reserved0` | - | Reserved |
| 2 | X2 | - | `txReady` | - | TX ready PIN configuration (see graphic below) |
| 4 | X4 | - | `mode` | - | A bit mask describing the UART mode (see graphic below) |
| 8 | U4 | - | `baudRate` | Bits/s | Baudrate in bits/second |
| 12 | X2 | - | `inProtoMask` | - | A mask describing which input protocols are active.<br>Each bit of this mask is used for a protocol. Through that, multiple protocols can be defined on a single port. (see graphic below) |
| 14 | X2 | - | `outProtoMask` | - | A mask describing which output protocols are active.<br>Each bit of this mask is used for a protocol. Through that, multiple protocols can be defined on a single port. (see graphic below) |
| 16 | U2 | - | `reserved4` | - | Always set to zero |
| 18 | U2 | - | `reserved5` | - | Always set to zero |

**Bitfield txReady**

This Graphic explains the bits of `txReady`

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| thres | | | | | | | | | pin | | | | | pol | en |

▢ signed value
🟩 unsigned value
▢ reserved

Let's assume we are not tweaking txReady bits yet, so our hand-crafted packet will look like this:

```
// Send a packet to the receiver to change baudrate to 115200.
void changeBaudrate() {
    // CFG-PRT
    byte packet[] = {
        0xB5, // sync char 1
        0x62, // sync char 2
        0x06, // class
        0x00, // id
        0x14, // length
        0x00, //
        0x01, // payload
        0x00, // payload
        0x00, // payload
        0x00, // payload
        0xD0, // payload
        0x08, // payload
        0x00, // payload
        0x00, // payload
        0x00, // payload
        0xC2, // payload
        0x01, // payload
        0x00, // payload
        0x07, // payload
        0x00, // payload
        0x03, // payload
        0x00, // payload
        0x00, // payload
        0x00, // payload
        0x00, // payload
        0x00, // payload

        0xC0, // CK_A
        0x7E, // CK_B
    };
    sendPacket(packet, sizeof(packet));
}
```

sendPacket() just writes bytes to HardwareSerial

```
void sendPacket(byte *packet, byte len) {
    for (byte i = 0; i < len; i++)
    {
        GPS.write(packet[i]);
    }
}
```

## There will be necessary to re-init HardwareSerial to use new baudrate:

```
GPS.begin(9600, SERIAL_8N1, 34, 12);   //default mode
  changeBaudrate();
  delay(100); GPS.flush();
GPS.begin(115200, SERIAL_8N1, 34, 12);   //new 115k boost mode
```

**Voila! Now we have much faster Serial — 12x faster, to use with our tweaked GPS measurements rate, which are already 10x faster than defaults.**

Hope that helps.

### Some additional notes, may be important:

Note 1: always **test your device if it actually capable of 115k**. NEO chip is still a chinese random especially when faked, and other schematic may differ between versions;

Note 2: if your device have battery power — just like T-Beam, for instance — please note that CPU chip reset (as well as RST, and IDE code upload' reset) **are not resetting NEO GPS module**, it's still on power and operational. In particular, data **options you set earlier** might affect your UART port rates.