# Mini-Crate Secondary Board Reference Manual

## MCSB Commands Description

**L. Castellani**

**I.N.F.N. sez. PADOVA**

**16 December 2014**

# Introduction

This board manages seven RS485 bus, six for mini-crates for the wheel sector and one for TSB, there is also a bus-1wire to which possible connect ADC DS2450 or temperature sensor DS18S20, two temperature sensors are present on the board. There are also one optical RS232 compatible with old board, a second optical RS232 for full-band communication at 230400 baud, and one electrical RS422 for bridging communication between two board. The board have an internal CAN bus with ten node, one per bus output. The RS422 have node number 0, the seven RS485 bus have node number from 1 to 7, the optical RS232 compatible have node number 8 and full-band optical RS232 have node number 9.

The seven RS485 bus and compatible optical RS232 use the mini-care secondary frame format and answer to the followings commands:

**<u>Status, code 0xEA:</u>**
Return the state of the node that manages the bus.

Return data:
```
      char 0xC4;
      char hver;       Firmware version H.
      char lvers;      Firmware version L.
      char node;       CAN bus node number.
      char hid485;     ID H
      char lid485;     ID L to which node answer on RS485 bus
      char hdest;      CAN node H
      char ldest;      CAN node L to witch RS485 data are rute(if -1 the RS485
                       data are not routed)
      int  adc[4];     four 10bits adc conversions (Vref~=4.7/4.8V).
                       Little-endian format.
                       adc[0]=VCC/2 of the left near node;
                       adc[1]=VCC/2 of the right near node;
                       adc[2]=RS485_N level for bus monitor;
                       adc[3]=RS485_P level for bus monitor.
                       For optical RS232:
                             adc[2]=Optical signal;
                             adc[3]=Board current (V=20*I*0.1).
                       For RS422:
                             adc[2]=unused;
                             adc[3]=Board current (V=20*I*0.1).
```

**<u>Sel Output, code 0xD3:</u>**
Select the output to which route RS485 data.

Arguments:
```
      char out;        Outputs from 0 to 6, that identifier a CAN node write on
                       alias table, if out is 255 the route is disabled, if out
                       is 254 the route is disabled and locked, to unlock send
                       out=255 and after a new valid out.
```
Return data:
```
      char 0xFC;
      char 0xD3;
      char error;      0=OK, 1=previous selected can node not answer,
                       2=selected can node not answer, -2=this node is locked.
```

**<u>CAN command, code 0xC5:</u>**
Send a command on CAN bus.
The 29bits identifier of the CAN frame are divide in 8bit as source node, 8bit as destination node, 8bit as frame number and 5bit as port number, besides on the bus is implemented an handshake for every frame.  WARNING the command codes from 0xC0 to 0xCF are used on CCB server. Use special code HOST  (0xEC 0xXX) at the beginning to resolve overlap.

Arguments:
```
      char reply;       0 or 1 to indicate if the CAN command generate a reply
                        CAN frame.
      char port;        port number.
      char dest;        destination node.
      char data[max 8]; data[0]=CAN command, data[1]..data[7] arguments.
                        The data can be less of 8bytes.
```

Return data on error or replay=0:
```
      char 0xFC;
      char 0xC5;
      char error;       0=OK, 1=error destination node not answer.
```

Return data replay=1:
```
      char 0xC6;
      char 0xC5;
      char data[max 8]; return data, can be less than 8bytes, the content depend
                        by send CAN command.
```

**<u>Read Optical Histogram, code 0xC8:</u>**
Read the histogram of the optical signal.
This command is valid only on optical compatible port.

Arguments:
```
      char reset;       Optional argument, if 1 reset histogram.
```
Return data:
```
      char 0xC9;
      char base;        the offset of the first histogram element.
      char histog[32];  the histogram bins
```

**<u>Special return data code 0xFC:</u>**
The codes that follow 0xFC code are:
```
0x00  unknown command
0xFF  routing disabled or locked (valid only for compatible optical RS232 node)
```

# CAN command over RS485 bus:

**Read Error, code 1:**

Read error information status.

```
Reply=1;
Port=0;
```

Arguments:

Return data from RS485 node:

```
      char CanTXErrCnt; CAN Transmit Error Count Register.
      char CanRXErrCnt; CAN Receive Error Count Register
      char CanErrCode;  First CAN error code
      int AckTime;      Time latency of the acknowledge (multiple frame mode),
                        t(ms)=AckTime*5. Little-endian format.
      int AckTime1;     Time latency of the acknowledge (single frame mode),
                        t(ms)=AckTime1*5. Little-endian format.
      char RsErrCode;   First RS error code.
```

Return data from node 0 and 9:

```
      char CanErrCode;  First CAN error code
      char RsErrCode;   First RS error code.
      char CanTXErrCnt; CAN Transmit Error Count Register.
      char CanRXErrCnt; CAN Receive Error Count Register
      char unused;      always 0
      char RSErrCnt;    RS Error Counter.
```

**Read CAN Error Counters, code 2:**

Read the CAN error counters.

```
Reply=1;
Port=0;
```

Arguments:

```
      char bank;        Bank number must be incremented up to return zero byte
                        or less than eight bytes.
```

Return data:

```
      char err[8];      the error counters;
```

Return data from RS485 nodes:

**CAN error counters description:**

| Counter | Description |
|---|---|
| 0 | Hardware buffer overflow |
| 1 | Command buffer overflow |
| 2 | TX Acknowledge buffer overflow |
| 3 | Port number error |
| 4 | Software error |
| 5 | Not received Acknowledge on TX data frame |
| 6 | Double RX data frame |
| 7 | Double RX data frame (frame already processed) |
| 8 | RX Acknowledge error |
| 9 | RX Acknowledge error (TX data frame not in FIFO) |
| 10 | RX Acknowledge error (TX buffer overflow) |
| 11 | RX Frame number error |
| 12 | TX frame repeated |

Return data node 0 and 9:

**CAN Error counters description:**

| Counter | Description |
|---|---|
| 0 | CAN Hardware buffer overflow |
| 1 | CAN TX buffer overflow |
| 2 | CAN Software error |
| 3 | CAN Command buffer overflow |
| 4 | CAN Local command Acknowledge error |
| 5 | CAN Not received Acknowledge on TX data frame |

## Destination, code 3:

Set destination CAN node of the data received on RS485 bus.
If destination is -1 (0xFFFF) or bigger than 255 the data are not routed.
To have bidirectional communications for RS485 data between two nodes, for example node 1 and node 2, the destination on node 1 must me set to 2 and destination on node 2 must be set to 1, and be sure that other node not use destination 1 or 2. The use of old value returned by command is deprecated to clear old destination after set new destination, use Get Destination command to clear connection between two node and after use Set Destination for new node connection.

```
Reply=1;
Port=0;
```

Arguments:

```
      char destL;       Low part of the destination node.
      char destH;       High part of the destination node.
```

Return data:

```
      char oldL;        Low part of previous destination node.
      char oldH;        High part of previous destination node.
```

## Reset, code 4:

Reset node. Return always an error on RS485 bus, because the node is reset without send acknowledge frame.

```
Reply=0;
Port=0;
```

## Set Destination, code 5:

Set destination CAN node of the data received on RS485 bus.
If destination is -1 (0xFFFF) or bigger than 255 the RS485 data are not routed.
To have bidirectional communications for RS485 data between two nodes, for example node 1 and node 2, the destination on node 1 must me set to 2 and destination on node 2 must be set to 1, and be sure that other node not use destination 1 or 2.

```
Reply=0;
Port=0;
```

Arguments:

```
      char destL;       Low part of the destination node.
      char destH;       High part of the destination node.
```

**Get Destination, code 6:**
Get destination CAN node of the data received on RS485 bus.
If destination is -1 (0xFFFF) or bigger than 255 the data are not routed.

```
Reply=1;
Port=0;
```
Arguments:
Return data:
```
      char destL;       Low part of previous destination node.
      char destH;       High part of previous destination node.
```

**Set ID, code 7:**
Set identifier of the RS485 bus. The value take effect after a reset

```
Reply=0;
Port=0;
```
Arguments:
```
      char idH;         High part of the RS485 identifier.
      char idL;         Low part of the RS485 identifier.
```

**Get ID, code 8:**
Get identifier of the RS485 bus.

```
Reply=1;
Port=0;
```
Arguments:
Return data:
```
      char idH;         High part of the RS485 identifier.
      char idL;         Low part of the RS485 identifier.
```

**Syncronize, code 9:**
Synchronize the CAN frame number. After a reset there are 1/256 probability that the first transmitted frame can be seen as already received frame and so ignored, repeat the command just one time on error.

```
Reply=0;
Port=0;
```

**Reset error counters, code 10:**
Reset all error counters except CAN Transmit/Receive Error Count Registers.

```
Reply=0;
Port=0;
```

**<u>Read RS Error Counters, code 11:</u>**
Read the RS error counters. This command is not valid on node 0 and 9.

```
Reply=1;
Port=0;
```
Arguments:

| | |
|---|---|
| `char bank;` | `Bank number must be incremented up to return zero byte or less than eight bytes.` |

Return data:

| | |
|---|---|
| `char err[8];` | `the error counters;` |

Return data from RS485 node:

**RS error counters description:**

| Counter | Description |
|---|---|
| 0 | Hardware Receive Buffer Overrun |
| 1 | Frame error |
| 2 | Software Receive buffer overflow |
| 3 | CRC error |
| 4 | Start of Frame error |
| 5 | Transmitter buffer overflow |

Return data node 0 and 9:

**RS error counters description:**

| Counter | Description |
|---|---|
| 0 | RS Hardware Receive Buffer Overrun |
| 1 | RS Frame error |
| 2 | RS Software Receive buffer overflow |
| 3 | RS Start of Frame error |
| 4 | RS Software error |
| 5 | RS Timeout error |
| 6 | RS CRC error |
| 7 | RS Command buffer overflow |

**<u>Set DAC, code 12:</u>**
Set 10bits DAC to manage optical receiver threshold and transmitters current laser, Vref ~= 4.7/4.8V.
This command is valid only on optical RS232/485 node.

```
Reply=0;
Port=0;
```
Arguments:

| | |
|---|---|
| `char ch;` | `ch=1 threshold, ch=2 laser current.` |
| `char dacL;` | `Low part of the 10bits DAC value.` |
| `char dacH;` | `High part of the 10bits DAC value.` |

**Read ADC DS2450, code 13:**

Read the 16bit ADC conversion. The value is update about every  second.
This command is valid only on CAN node 7.

```
Reply=1;
Port=0;
```

Arguments:

```
      char nadc;        adc number, if adc is not present return zero byte
```

Return data:

```
      char ch_a_L;      Low part of the channel A conversion.
      char ch_a_H;      High part of the channel A conversion.
      char ch_b_L;      Low part of the channel B conversion.
      char ch_b_H;      High part of the channel B conversion.
      char ch_c_L;      Low part of the channel C conversion.
      char ch_c_H;      High part of the channel C conversion.
      char ch_d_L;      Low part of the channel D conversion.
      char ch_d_H;      High part of the channel D conversion.
```

**Read ADC ID DS2450, code 14:**

Read ADC identifier.
This command is valid only on CAN node 7.

```
Reply=1;
Port=0;
```

Arguments:

```
      char nadc;        adc number, if adc is not present return zero byte
```

Return data:

```
      char id[8];       the 64bit device identifier.
```

**Initialize B1W, code 15:**

Initialize bus 1-wire and search ADCs DS2450 and temperature sensors DS18S10, max four device each type. This command is valid only on CAN node 7.

```
Reply=1;
Port=0;
```

Arguments:

Return data:

```
      char error;       The DS2482 initialize result, 0=OK.
      char nadc;        The number of DS2450 found.
      char ntsens;      The number of DS18S20 found.
```

**Read Temperature DS18S20, code 16:**

Read the ADC conversion. The value is update about every second.
This command is valid only on CAN node 7.

```
Reply=1;
Port=0;
```

Arguments:

```
      char nsens;       The sensor number, if the sensor is not present return
                        zero byte.
```

Return data:

```
      float temp;       The converted temperature, 3byte format equivalent to
                        4byte IEEE32 little endian format with first byte=0
```

**Read Temperature sensor ID DS18S20, code 17:**

Read temperature sensor identifier.
This command is valid only on CAN node 7.

```
Reply=1;
Port=0;
```

Arguments:

```
      char nsensor;     Sensor number, if sensor is not present return zero byte
```

Return data:

```
      char id[8];       The 64bit device identifier.
```

**Set Alias Table, code 18:**

Set the alias table. The table contain the CAN node number for the SelOutput command on RS485 bus.

```
Reply=0;
Port=0;
```

Arguments:

```
      char sel0_node;   The node number used with SelOutput=0.
      char sel1_node;   The node number used with SelOutput=1.
      char sel2_node;   The node number used with SelOutput=2.
      char sel3_node;   The node number used with SelOutput=3.
      char sel4_node;   The node number used with SelOutput=4.
      char sel5_node;   The node number used with SelOutput=5.
      char sel6_node;   The node number used with SelOutput=6.
```

**Read ADC Internal, code 19:**

Read the 10bit ADC internal to the node, Vref~=4.7/4.8V.

```
Reply=1;
Port=0;
```

Arguments:

```
      char ch;          The physical channel number of the adc, see schematics
```

Return data:

```
      char adcL;        Low part of the ADC.
      char adcH;        High part of the ADC.
```

**Read Optical Histogram, code 20:**

This command is valid only on optical node and permit to read the optical signal amplitude.
On the overflow the histogram bins are divided by two.

```
Reply=1;
Port=0;
```

Arguments:

```
      char bank;        the bank to read.
      char reset;       optional argument, if 1 reset the histogram.
```

Return data bank=0:

```
      char base;        the offset of the first histogram element .
      char histog[7];
```

Return data bank!=0:

```
      char histog[sz];  if the sz is less than 8 there aren't more data.
```

**Latchup code 21:**

Get info or active mosfet for latch up protections. Latch up protection short-circuit the power supply on the near PIC node.
The latch up monitor watch the power supply of the near PICs and if it is less than a 1V (poliswitch active) for more than 0.5 seconds short-circuit the power with a MOSFET for 5 seconds, and after if the power supply don't return to the normal value the monitor is disabled.

```
Reply=1;
Port=0;
```
Arguments:
```
      char ch;    ch=0 read the state.
                  ch=1 active latch up mosfet for 0.6sec on left PIC;
                  ch=2 active latch up mosfet for 0.6sec on right PIC;
                  ch=11 active latch up mosfet on left PIC;
                  ch=12 active latch up mosfet on right PIC;
                  ch=10 disable latch up mosfet on left and right PIC;
                  ch=255 reset counters and state machine and reactivate the
                  monitor if disabled;
```
Return data:
```
      char counter1;    Left counter for the latch up event.
      char counter2;    Right counter for the latch up event.
      char flags;       bit0 left monitor disabled;
                        bit1 right monitor disabled;
                        bit2 left event in process;
                        bit3 right event in process.
```

**Version code 22:**

Get firmware version.

```
Reply=1;
Port=0;
```
Arguments:

Return data:
```
      char versionL;    Low part of the firmware version.
      char versionH;    High part of the firmware version.
```

**GetDAC code 23:**

Get threshold and laser dac setting, valid only for optical node 8 and 9.

```
Reply=1;
Port=0;
```
Arguments:

Return data:
```
      char thrL;        Low part of the threshold.
      char thrH;        High part of the threshold.
      char laserL;      Low part of the laser.
      char laserH;      High part of the laser.
```

**Get Alias Table, code 24:**
Get the alias table. The table contain the CAN node number for the SelOutput command on RS485 bus. This command is valid only on node 8.

```
Reply=1;
Port=0;
```
Arguments:
Return data:
```
      char sel0_node;   The node number used with SelOutput=0.
      char sel1_node;   The node number used with SelOutput=1.
      char sel2_node;   The node number used with SelOutput=2.
      char sel3_node;   The node number used with SelOutput=3.
      char sel4_node;   The node number used with SelOutput=4.
      char sel5_node;   The node number used with SelOutput=5.
      char sel6_node;   The node number used with SelOutput=6.
```

# Full-Band Frame

The frame format on full-band optical RS232 and electrical RS422 is:

```
[SOF] //start of frame = 0x55
      //    7     6     5      4      3      2      1      0
[sIDh] //[SID10][SID9] [SID8] [SID7] [SID6] [SID5] [SID4] [SID3] Source
[sIDl] //[SID2] [SID1] [SID0] [-]    [EXIDE][-]    [EID17][EID16]Port
[eIDh] //[EID15][EID14][EID13][EID12][EID11][EID10][EID9] [EID8] Destination
[eIDl] //[EID7] [EID6] [EID5] [EID4] [EID3] [EID2] [EID1] [EID0] nframe
[dlc]  //[-]    [RTR]  [-]    [-]    [SZ3]  [SZ2]  [SZ1]  [SZ0]  size data o RTR
[data 0] //data
.
.
[data n-1] //n=SZ[0..3] (max 8) see dlc.
[crc8]
```

The node work as a RS232-CAN modem,  the frame excluded SOF and CRC8, are the CAN controller internal registers. All frame received corrected on RS232 are transferred to the CAN bus, so all frame received on CAN bus are transferred to the RS232 bus, except in case of  buffer overflow or reception error .

The protocol implemented on RS485 node foresee that any data frame (RTR =0) sent, need an acknowledge return frame that is identical to sent frame except that have not data and RTR bit is set to 1 (CAN Remote Frame is the acknowledge frame of the protocol). The 29bit CAN identifier are divide as show above,  note that nframe must be incremented by one for successive frame to the same destination. If  acknowledge is not received, the frame must be retransmitted for maximum 3 time after 300ms interval.

Node number from 0 to 15 are reserved for physical node, the rest can be used for virtual node in the server.

Port number is 5bits (`[SID2][SID1][SID0][EID17][EID16]`) and EXIDE bit must be set to 1.

In the figure 1 is represented the structure of a server

On the RS485 node are implemented four port:

Port 0, for the control command describe above

Port 3, for the reply of the control command.

This ports support single frame transaction from node to node, so you must wait acknowledge frame before to send new one at the same node. Different node can send command to the same node asynchronous.

Port 1, for RS485 space data parity (mini-crate data frame) .

Port 2, for RS485 mark data parity  (mini-crate address frame).

This ports support multiple frame transaction, up to 32 frame, from node to node, the transmission order on RS485 bus is guaranteed by frame number also if frames are lost and retransmitted. Different node can’t send data to the same node.

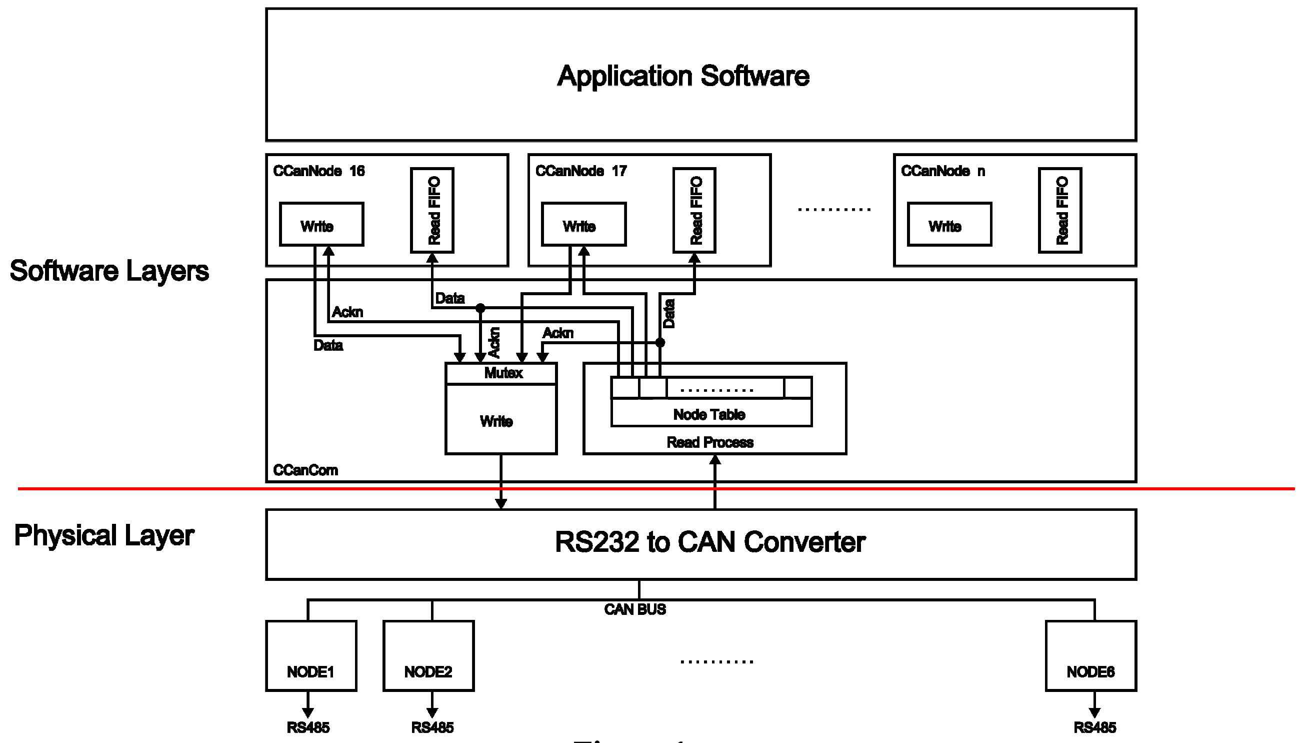

**Figure 1**

To communicate with full-band node simply use its node number.
On the nodes are implemented on Port=0 the command describe in “CAN Command over RS485 bus” paragraph.
The return data have Port value set to 3.
To Communicate by CAN bus through RS485 node you must use CAN command, code 0xC5, or by other full-band node.

## McSecSever Software

This software implement the structure show in the figure 1, to communicate with minicrate, and also it implement a TCP sever that create a virtual node (CCanNode) for every client connection, that can be used to monitor the McSecondaryBoard.
The virtual node to work properly, after having established the connection, must be initialized.
The number of connections is physically limited to about 40 by bits assigned to identifies the node and by node used/mapped by software.

The frame accepted by TCP connections is defined as :

```
typedef struct {
      TCPHEADER header;
      TCPCANFRAME frame[64];
} TCPFRAMEBUFFER;

typedef struct {
      DWORD sof;
      DWORD nframe;
      DWORD type;
} TCPHEADER;
```

```
typedef union {  //warning sizeof(TCPCMD)<=sizeof(CANFRAME)
      CANFRAME can;
      TCPCMD   tcp;
} TCPCANFRAME;

typedef struct {    //    7     6     5      4     3     2     1     0
      BYTE sIDh;  //[SID10][SID9] [SID8] [SID7] [SID6] [SID5] [SID4] [SID3]     Source
      BYTE sIDl;  //[SID2] [SID1] [SID0] [-]    [EXIDE][-]    [EID17][EID16]    Port
      BYTE eIDh;  //[EID15][EID14][EID13][EID12][EID11][EID10][EID9] [EID8]     Destination
      BYTE eIDl;  //[EID7] [EID6] [EID5] [EID4] [EID3] [EID2] [EID1] [EID0]     nframe
      BYTE dlc;   //[-]    [RTR]  [-]    [-]    [SZ3]  [SZ2]  [SZ1]  [SZ0]      size data o RTR
      BYTE data[8];
} CANFRAME;

typedef struct {
      BYTE cmd;
      BYTE arg[12];
} TCPCMD;
```

The `sof`:

```
#define TCPSOF    0x5555AAAA
```

The `nframe` is the number of `TCPCANFRAME` transmitted;

The `type`:

```
 enum TYPEFRAME {
      TYPE_CAN,
      TYPE_TCP
};
```

The `cmd`:

```
enum TCP_CMD {
      TCPCMD_ASSIGNMODE,
      TCPCMD_CMDOK,
      TCPCMD_ACKERROR,
      TCPCMD_CMDERROR,
};
```

## Initialize the Node

After having established the connection the node must be initialized with the following command:

```
TCPFRAMEBUFFER tcpframe;
tcpframe.header.sof=TCPSOF;
tcpframe.header.type=TYPE_TCP;
tcpframe.header.nframe=2;

tcpframe.frame[0].tcp.cmd=TCPCMD_ASSIGNMODE;
tcpframe.frame[0].tcp.arg[0]=PORT_CMD;
tcpframe.frame[0].tcp.arg[1]=SINGLE_FRAME;

tcpframe.frame[1].tcp.cmd=TCPCMD_ASSIGNMODE;
tcpframe.frame[1].tcp.arg[0]=PORT_REPLAY;
tcpframe.frame[1].tcp.arg[1]=SINGLE_FRAME;
```

Below the constant definitions:

```
#define PORT_CMD        0
#define PORT_RS485S     1
#define PORT_RS485M     2
#define PORT_REPLAY     3
#define PORT_DATA       0x1F
#define PORT_ADDR       0x1E
#define PORT_READ       0x1D

enum RXMODE{
            IGNORE_FRAME,
            SINGLE_FRAME,
            MULTIPLE_FRAME
      };
```

This two command initialize the node to send and receive only the CAN frame to monitor the board.

**The return data from server is**:

```
tcpframe.header.sof             == TCPSOF
tcpframe.header.type            == TYPE_TCP
tcpframe.header.nframe          == 2
tcpframe.frame[0].tcp.cmd       == TCPCMD_CMDOK
tcpframe.frame[1].tcp.cmd       == TCPCMD_CMDOK
```

Now can be send the CAN frame to communicate with the physical node on the board

```
tcpframe.header.sof=TCPSOF;
tcpframe.header.type=TYPE_CAN;
tcpframe.header.nframe=1;

port=PORT_CMD;
dest=1;     //destination node
size=2;     //size of can data

tcpframe.frame[0].can.sIDh = 0; //override by server
tcpframe.frame[0].can.sIDl = (port & 3) | ((port & 0x1C)<<3) | 0x08;
tcpframe.frame[0].can.eIDh = dest;
tcpframe.frame[0].can.eIDl = 0; //override by server
tcpframe.frame[0].can.dlc  = size;
tcpframe.frame[0].can.data[0] = 2; //command code
tcpframe.frame[0].can.data[1] = 0; //command argument
```

From the server return one acknowledge can-frame to indicate that the write frame is received by destination node and an eventually one replay can-frame, it depend by command.

**The acknowledge can-frame**:

```
tcpframe.header.sof             == TCPSOF
tcpframe.header.type            == TYPE_CAN
tcpframe.header.nframe          == 1
tcpframe.frame[0].can.sIDl      == (port & 3) | ((port & 0x1C)<<3) | 0x08;
tcpframe.frame[0].can.eIDh      == dest;
tcpframe.frame[0].can.dlc       == 0x40;
```

**or in case of write error**:

```
tcpframe.header.sof            == TCPSOF
tcpframe.header.type           == TYPE_TCP
tcpframe.header.nframe         == 1
tcpframe.frame[0].tcp.cmd      == TCPCMD_ACKERROR
```

**The reply can-frame**:

```
tcpframe.header.sof         == TCPSOF
tcpframe.header.type        == TYPE_CAN
tcpframe.header.nframe      == 1
tcpframe.frame[0].can.sIDh //the source node
tcpframe.frame[0].can.sIDl == (PORT_REPLAY & 3)|((PORT_REPLAY & 0x1C)<<3)|0x08;
tcpframe.frame[0].can.eIDh //this node number assigned by server
tcpframe.frame[0].can.eIDl //frame number
tcpframe.frame[0].can.dlc  //size of data valid
tcpframe.frame[0].can.data[0] //data
..
tcpframe.frame[0].can.data[7] //data
```

For the moment is discourage to send multiple can-frame with one tcp-frame, future modification of the server will be done, but it can be done, in this case you receive only one acknowledge frame and can receive multiple replays in one or more tcp-frame and in any order.
It is also deprecated send multiple can-frame to the same node because the node have the FIFO by only  tree message, instead can be send multiple can-frame to different node.