FPC

MATRIXBOX MX2400 SERIES

System Manual

Revision 2018.2.15

FPC LLC
1401 21ST ST STE R
SACRAMENTO, CA 95811

E-mail: info@funtestfpc.com
Tel.: 213-431-9776

Table of Contents

1 Introduction 4

2 Communication 4

2.1 Ethernet . . . 4
2.1.1 Control channel . . . 4
2.1.2 Event channel . . . 4
2.1.3 Packet format . . . 5
2.1.4 Answer . . . 5
2.1.4.1 Return codes . . . 5
2.1.5 Keep alive . . . 7
2.2 RS-232 . . . 7
2.2.1 Activation . . . 7
2.2.2 Communication format . . . 8

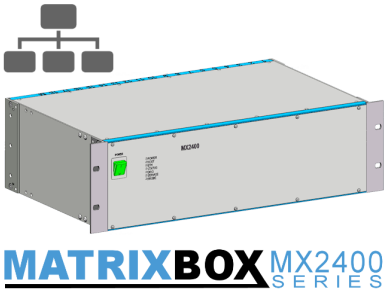
3 Commands 8

3.1 sys (System) . . . 8
3.1.1 *idn? (Identification) . . . 8
3.1.2 rtc|rtc? (Real-time clock) . . . 9
3.1.3 fs (File-system) . . . 9
3.1.4 file (File operations) . . . 11
3.2 net (Network) . . . 12
3.2.1 mac? (Read MAC address) . . . 12
3.2.2 ip? (Read configured IP address) . . . 12
3.2.3 mask? (Read configured IP mask) . . . 13
3.3 io (Input/outputs) . . . 13
3.3.1 out (Set outputs) . . . 13
3.3.2 out? (Query outputs) . . . 13
3.3.3 in? (Query inputs) . . . 14
3.3.4 Event channel . . . 14
3.4 card (Card system) . . . 14
3.4.1 *detect (Card detection) . . . 14
3.4.2 detect? (List of detected cards) . . . 15
3.4.3 *rst (Cards reset) . . . 15
3.4.4 cnt? (Card count) . . . 15
3.4.5 *idn? (Card identificaton) . . . 15
3.4.6 ver? (Card firmware version) . . . 16
3.4.7 led (Card status LED control) . . . 16
3.5 mx (Matrix) . . . 16
3.5.1 clr|set|cset (Test-point control) . . . 17
3.5.2 route|croute (MX card routing) . . . 17
3.5.3 route|croute (Master routing) . . . 17
3.6 mxq (MX system queries) . . . 18

3.6.1 tp? (TP states query) . . . 18
3.6.2 diag (Diagnostics query) . . . 18
3.6.3 route? (Routing state query) . . . 19
3.7 probe (TP probe) . . . 20
3.7.1 start (Start probe) . . . 20
3.7.2 stop (Stop probe) . . . 21
3.7.3 active? (Check probe status) . . . 21
3.7.4 Event channel . . . 21
3.8 mm (Multimeter) . . . 22
3.8.1 route (Set routing) . . . 22
3.8.2 route? (Query routing) . . . 23
3.8.3 meas? (Perform a measurement) . . . 23
3.8.4 zero (Set zero) . . . 24
3.8.5 *rst (Reset) . . . 25
3.9 fp (Front panel) . . . 25
3.9.1 led (LED control) . . . 25
3.10 dio (DIO card control) . . . 26
3.10.1 in? (Read inputs) . . . 26
3.10.2 out? (Read outputs) . . . 26
3.10.3 out (Write outputs) . . . 27
3.10.4 clra (Clear all outputs) . . . 27
3.10.5 trig (Output trigger) . . . 27
3.10.6 count? (Channel count) . . . 27
3.10.7 cm (Set common config) . . . 28
3.10.8 cm? (Query common config) . . . 28
3.10.9 diag (Run diagnostics) . . . 28
3.10.10 diag? (Read diagnostics result) . . . 29
3.10.11 Event channel . . . 29
3.11 bct (Batch-test) . . . 29
3.11.1 clr (Clear TP list) . . . 29
3.11.2 load|cload|load? (Load TP list) . . . 30
3.11.3 syst|syst? (Measurement system) . . . 30
3.11.4 print? (Prints TP list) . . . 31
3.11.5 res? (Measurement results) . . . 32
3.11.6 run|run? (Run batch-test) . . . 32
3.11.7 abort (Abort batch-test) . . . 33

1 Introduction

In this system manual the low-level communication of the MX2400 system is described. This allows the integration to third-party software solutions.



2 Communication

The communication with the host system via ethernet is based on TCP protocol. Only one host system can be connected to the MX2400 at one time.

There are two TCP channels:
- control channel
- event channel

Each channel has a specified functionality and listen on dedicated TCP ports. Both act like server, the host system connects to them via the client.

2.1 Ethernet

The main connection of MX2400 to the host system.

2.1.1 Control channel

Control channel is used at most of the time to send request and receive answer from MX2400.

Features

- TCP port: 2400
- Fully synchronous
- Two-way (request/answer)
- 2kB packet size limit

2.1.2 Event channel

Event channel is used to send and information from MX2400 to the host system.

Features

- TCP port: 2401
- Asynchronous
- One-way only (from the MX2400 -> host system)
- Keep-alive functionality
- 2kB packet size limit

2.1.3 Packet format

Nomenclature

- master - the MX2400 master card (TCP server)
- host - typically PC (TCP client)

The request and answer is packet based, each packet has defined end by the ending character. When begin, the master receives the data as long as the ending character is received.

Full packet format

```
<header><HE><message><ME>
```

- `header` - request header, set of simple named arguments with values, separated by a colon character
- `HE` - header ending character, fixed character \x01 (ASCII 0x01)
- `message` - request message, command text
- `ME` - message ending character, fixed character \x00 (ASCII 0x00 NUL)
- Header and message is plain-text format (ASCII 32 to 126), no special characters cannot be used here.

Header arguments and message depends on specific command. The format is the same for both control and event channel. Commands which require to transfer a binary data uses base-64 encoding.

Header format

$: arg_0 = value_0 : arg_1 = value_1 : ... : arg_N = value_N$

- arg_N - name of argument
- $value_N$ - value of argument

Request

Each request sequence starts by sending the command to the host and ends by receiving the answer from the host. The request is blocking, another request is not possible until currently processed one is finished. Host system must always wait for an answer from the master within specific time, generally 3000 ms.

2.1.4 Answer

Because the communication is request-answer type, on each request the master sends the answer back to the host. The answer has the same format like request - header, message and endings.

2.1.4.1 Return codes

The answer has a persistent header argument - **return code**. The return code is **numerical result** of processed command.

```
: rc=<code>
```

Return code types

- `0x200` - OK
- `0x300` - warning
- `0x400` - error

Warning codes

Warning uses the bit-mask to encode more than one warning in one code (8 maximum).

Code	Name	Description
0x300 + 0x01	WARN_EXCEED_NR_OF_ARGS	Number of arguments exceeded (limit: 32), arguments over 32 are discarded
0x300 + 0x80	WARN_NO_CARD_ACCEPTED_CMD	No card in the system receives the broadcast command

Error codes

Code	Name	Description
0x401	ERR_WRONG_MSG_FMT	Wrong message format received over the socket, see full packet format
0x411	ERR_HDR_REQUIRED	Specified header required
0x412	ERR_HDR_UNKNOWN	Unknown header
0x413	ERR_HDR_INVALID	Required header specified, but value is invalid
0x421	ERR_CMD_REQUIRED	Specified command required
0x422	ERR_CMD_UNKNOWN	Unknown command
0x423	ERR_CMD_INVALID	Command specified, but invalid
0x431	ERR_ARG_REQUIRED	Command's argument required
0x432	ERR_ARG_UNKNOWN	Unknown argument of command
0x433	ERR_ARG_INVALID	Command's argument specified, but with unexpected value
0x434	ERR_ARG_MUST_BE_NUMBER	Similar to INVALID, specify that a number must be passed
0x435	ERR_ARG_UNEXPECTED	Similar to INVALID
0x441	ERR_INVALID_OPERATION	Specified operation is invalid and cannot be processed
0x442	ERR_DENIED_OPERATION	Similar to INVALID_OPERATION
0x443	ERR_INVALID_DATA	Used when firmware updating, typically thrown when verify failed
0x450	ERR_FILE_ACCESS	File-system access error (e.g. file is not open, no free space, etc..)
0x481	ERR_CARD_DOES_NOT_EXIST	Card does not exist in the list of detected cards
0x482	ERR_CARD_REMOVED	Card was previously detected, but currently it is not in the system - re-detection is required
0x483	ERR_NO_TARGET	Card is not in the system
0x484	ERR_UNSUPPORTED_TARGET_TYPE	The command requires specified type of card
0x485	ERR_TARGET_SPECIFY_REQUIRED	The command requires to specify the target card
0x491	ERR_SYSTEM_BUS_BUSY	System bus is busy (any communication is on-going)
0x492	ERR_SYSTEM_BUS_TIMEOUT	Specified card did not respond within specified time
0x493	ERR_SYSTEM_BUS_CRC	The communication was done, but incoming data have incorrect CRC (corrupted data)
0x494	ERR_SYSTEM_BUS_ERR_SIGNAL	Any card indicates error via the dedicated ERROR signal

The master also sends the human-readable error message.

Software handling

Generally, because the master also sends the error message, it is not necessary to implement each warning and error code separately. It is sufficient to check the return code of answer and when returns 200 - OK, when 300..399 - do nothing (or make some log) or when 400..499 - throw an error with provided error message.

2.1.5 Keep alive

There is keep-alive functionality, which consists of special service packets to check the communication channel is alive and functional. This communication is done **over the event channel**.

The packet is 1-byte length only, without header, message and endings.

Every 1 second, the master sends to the host the KEEP_ALIVE_REQUEST. The host must response by KEEP_ALIVE_RESPONSE. If there is **inactivity for more than 5 seconds**, the master automatically closes the connection.

- KEEP_ALIVE_REQUEST = `0x07` (ASCII BELL)
- KEEP_ALIVE_RESPONSE = `0x06` (ASCII ACK)

2.2 RS-232

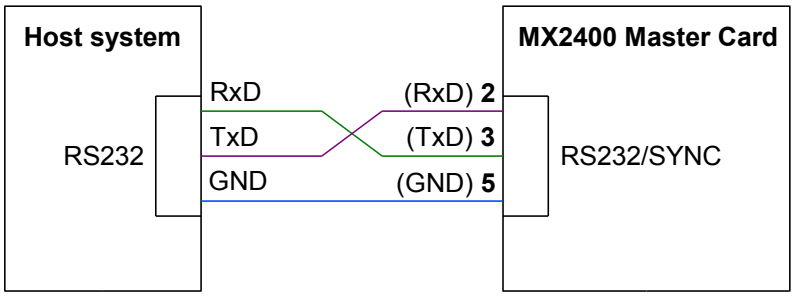
The secondary connection. RS-232 is typically intended for Service/SYNC purposes only, but it is possible to use it like an alternative control.

Features

- 115,2 kbit/s baudrate, 8 data bits, 1 stop bit, no parity, no flow control
- Request/answer communication
- No events supported

Connection to host

- Standard RS-232 female-female cross-link cable should be used



Minimum required interconnection between host system and MX2400 master card

RS232/SYNC is the CAN 9 connector on the Master Card. For details, see the pinout in the MX2400 Hardware Manual. It is possible to use a standard manfactured full-wired CAN9/CAN9 (with Rx/Tx crossed) cable, the pinout is designed to avoid conflicts.

If the SYNC functionality is required, then the special cable must be made to separate RS232 and SYNC signals.

2.2.1 Activation

Because the RS-232 is normally intended for SERVICE/SYNC usage, the control of MX2400 must be activated first by special packet.

Activation data sequence

The sequence consist of specified 4 bytes.

```
0xC0 0xC3 0xCC 0xCF
```

Return value

`#SMC\n`
These 5 bytes are sent like a confirmation that the control mode over RS232 has been activated.

This sequence can by sent anytime from the host system to the MX2400 master card and will cause activation the control mode.

2.2.2 Communication format

Request

```
\xAA<header>\xA0<message>\n
```

- `\xAA` - packet start-byte
- `header` - request header, set of simple named arguments with values, separated by a colon character
- `\xA0` - header ending byte
- `message` - request message, command text
- `\n` - packet ending byte (ASCII 10)

Answer

```
[<rc>{:<header>}]<message>\n
```

- `rc` - return code, identical to ethernet return codes, hexadecimal number without "0x" prefix
- `header` - optional return headers
- `message` - return message
- `\n` - packet ending byte (ASCII 10)

3 Commands

MX2400 has a number of sub-systems.

Each sub-system is represented by specific header and according commands. These commands are valid only with specified header.

3.1 sys (System)

System commands.

```
Header
f=sys
```

3.1.1 *idn? (Identification)

```
Command
*idn?
```

Return product identification.

Parameters

No parameters

Return value

`FPC s.r.o.,MatrixBox MX2400,<serial-number>,<firmware-version>`

3.1.2 rtc|rtc? (Real-time clock)

```
Command
rtc:<date-time>
```

Sets current date/time of the system.

Parameters

`date-time`	[string]	A date/time to set. Format: `YYYY-mm-dd`**`T`**`HH-mm-ss`

Return value

No return value

```
Command
rtc?
```

Returns current date/time of the system.

Parameters

No parameters

Return value

`YYYY-mm-dd`**`T`**`HH-mm-ss`

- `YYYY`: year 2000..2099
- `mm`: month 01..12
- `dd`: day 01..31 (depends on month)
- `T`: the date/time separator character
- `HH`: hour 00..23
- `mm`: minute 00..59
- `ss`: second 00.59

3.1.3 fs (File-system)

Internal file-system commands.

The MX2400 has a small internal FLASH memory with dedicated **256 KB user space**. This memory must be used to store the firmware when card update is about to do.

```
Command
fs:list?{:<last-page>}
```

List all files. The method must be called repeatedly until `*END` is sent.

Usage:

1. send `fs:list?` without any arguments
2. check for `*END`, if yes => file listing finished

3. parse result, save file to list
4. send `fs: list?` with last-page from previously parsed arguments
5. go to step 2

Parameters

`last-page`	[int]	Page for next start. Pass the value from previous call. **Default:** first page of file-system

Return value

`<last-page>: <name>: <desc>: <size>`

- `last-page` [int] - value for next fs:list? call
- `name` [string] - file name, up to 4 letters
- `desc` [string] - file description, up to 31 letters
- `size` [int] - file size in bytes

`*END`
When the file-listing is finished.

```
Command
fs: free?
```

Get free memory space.

Parameters

No parameters

Return value

Number of free bytes

```
Command
fs: capacity?
```

Get total memory space.

Parameters

No parameters

Return value

Memory capacity, in bytes.

```
Command
fs: erase
```

Erase the whole memory. This will take about 5 seconds.

Parameters

No parameters

Return value

No return value

3.1.4 file (File operations)

All files are stored in the file-system. This chapter is related to previous fs chapter.

```
Command
file:new:<name>:<description>
```

Create a new file in the file-system with specified name and description.

Parameters

name	[string]	File name to create, up to 4 letters. Do not use special letters or space. Use uppercase/lowercase letters and numbers only.
description	[string]	File description, up to 31 letters. Use standard ASCII characters from 32 to 126.

Return value

New file handle (integer number).

Example

```
file:new:"HCMX":"HCMX card firmware"
```

Create a new file "HCMX" with specified description.

```
Command
file:del:<name>
```

Delete file with specified name from the file-system.

Parameters

name	[string]	File name to delete.

Return value

No return value

```
Command
file:open:<name>
```

Open an existing file from the file-system and set the position of file to zero.

Parameters

name	[string]	File name to open.

Return value

New file handle (integer number).

```
Command
file:read:<handle>:<length>
```

Reads data from open file. The position of file internally increases by length number.

Parameters

handle	[int]	Handle of file to be read. This handle you can get by file:open

		command.
length	[int]	The batch of data to read, up to 128 bytes.

Return value

Base-64 encoded data.
*EOF key-string is appended at the end of encoded data when the end of file is reached. Check for this string at the end of data and strip it from data if necessary.

```
Command
file:write:<handle>:<data>
```

Writes data to open or newly created file. The position of file after writing is automatically set at the end of file.

Parameters

handle	[int]	Handle of file to write.
data	[string]	Base-64 encoded data, up to 128 bytes (172 letters of base-64 string).

Return value

No return value.

3.2 net (Network)

Network commands.

```
Header
f=net
```

3.2.1 mac? (Read MAC address)

```
Command
mac?
```

Read PHY MAC address.

Parameters

No parameters

Return value

MAC address in format:
aa:bb:cc:dd:ee:ff

3.2.2 ip? (Read configured IP address)

```
Command
*ip?
```

Read currently configured IP address of MX2400 system.

Parameters

No parameters

Return value

IPv4 address in format:
`a.b.c.d`

3.2.3 mask? (Read configured IP mask)

```
Command
mask?
```

Read currently configured IP mask of the system.

Parameters

No parameters

Return value

Mask address in format:
`a.b.c.d`

3.3 io (Input/outputs)

Internal DIO (16 IN, 16 OUT) manipulation.

```
Header
f=io
```

3.3.1 out (Set outputs)

```
Command
out:<value>
```

Set OUTPUT states.

Parameters

`value`	[int]	16-bit number, representing OUT0 to OUT15. Hexadecimal format, 4-place. LSB = OUT0, MSB = OUT15 1 = on, 0 = off

Return value

No return value

Example

`out:f00f`
Set OUT0..OUT3 and OUT12 to OUT15 on

3.3.2 out? (Query outputs)

```
Command
out?
```

Query OUTPUT states.

Parameters

No parameters

Return value

16-bit number, representing OUT0..OUT15, LSB = OUT0. Hexadecimal format, 4-place.

3.3.3 in? (Query inputs)

```
Command
in?
```

Read INPUT states.

Parameters

No parameters

Return value

16-bit number, representing IN0..IN15, LSB = IN0. Hexadecimal format, 4-place.

3.3.4 Event channel

IO input states are sent asynchronously via event channel. This is done only when changed is detected on inputs.

```
Header
f=io
```

Message data

```
Message
<states>
```

`states`: 16-bit number, representing IN0..IN15, LSB = IN0. Hexadecimal format, 4-place.

3.4 card (Card system)

Card manipulation and basic operations (detection, version info, ..).

```
Header
f=card
```

3.4.1 *detect (Card detection)

```
Command
*detect
```

Run card detection in the system. The function blocks until the detection is finished.

Parameters

No parameters

Return value

No return value

3.4.2 detect? (List of detected cards)

```
Command
detect?
```

Returns list of detected cards in the system after *detect command.

Parameters

No parameters

Return value

<addr0>,<type0>{:<addr1>,<type1>:...:<addrN>,<typeN>}

- addr: card address, zero based
- type: numerical card type, 139 = LCMX, 144 = DEV, 167 = HCMX, 200 = DIO

When no cards in the system: "-"

3.4.3 *rst (Cards reset)

```
Command
*rst
```

Reset all cards in the system.

Parameters

No parameters

Return value

No return value

3.4.4 cnt? (Card count)

```
Command
cnt?
```

Number of detected cards in the system.

Parameters

No parameters

Return value

Number of cards in the system (int)

3.4.5 *idn? (Card identificaton)

```
Additional header
a=<address>
```

```
Command
*idn?
```

Returns identification string of specified card by address.

Parameters

No parameters

Return value

Card identification string

3.4.6 ver? (Card firmware version)

```
Additional header
a=<address>
```

```
Command
*ver?
```

Returns version string of specified card.

Parameters

No parameters

Return value

Card version string

3.4.7 led (Card status LED control)

```
Additional header
a=<address>
```

```
Command
led: <status>
```

Card's STATUS LED control.

Parameters

`status`	[enum]	LED status: • `off` • `on`

Return value

Card version string

3.5 mx (Matrix)

Test-point matrix commands.
For detail information about routing - see the MX2400 Programmer's Manual, section Matrix - Routing.

```
Header
f=mx
```


3.5.1 clr|set|cset (Test-point control)

```
Command
clr:
set:
cset:
      <low>:<high>
      -or-
      L:<L1>{:<L2>:...:<LN>}:H:<H1>{:<H2>:...:<HN>}
```

Connect (`set`, `cset`) or disconnect (`clr`) a pair of TP to/from LOW/HIGH or a set of TP to/from LOW and set of TP to/from HIGH bus. The "cset" does almost the same like "set", but "cset" disconnect all previously connected TPs first.

For details, see MX2400 Programmer's Manual, section Matrix - Commands - clr|set|cset (Test-point control).

3.5.2 route|croute (MX card routing)

```
Command
route:
croute:<loc>{:<card-bus>}:<main-bus>
```

Interconnection of internal MX card's buses (or lines) to any main bus 1 - 4. The "croute" clears all previous routing before applying new (like cset/set of test-point control). The "route" clears only settings of passed main-bus.

For details, see MX2400 Programmer's Manual, section Matrix - Commands - route|croute (MX card routing).

3.5.3 route|croute (Master routing)

```
Command
route:
croute:<loc>{:<card-bus>}:<main-bus>
```

Interconnection of main bus and Connection card input/output signals

```
Command
route:
croute:ps{:<ps-nr>:<state>}
```

External power supplies ON/OFF control.

```
Command
route:
croute:imeas{:<imeas-nr>|off}
```

External current measurement control. During switching the measurement, the internal bypass realy is automatically closed to prevent supply interruption.

```
Command
croute:m
```

Clear all blocks of master routing (mux, ps, imeas). The "`m`" parameter is required to distinguish the "croute" command for MX card routing and master routing.

For details, see MX2400 Programmer's Manual, section Matrix - Commands - route|croute (Master

routing).

3.6 mxq (MX system queries)

Matrix system query commands.

```
Header
f=mxq: a=<address>
```

a	[int]	Card address, zero-based

3.6.1 tp? (TP states query)

```
Command
tp?
```

Get current TP states of specified MX card.

Parameters

No parameters

Return value

<low>: <high>: <tp_0><tp_1>...<tp_N>

low, high	[int]	TP range of specified card
tp	[enum]	TP states: • -: none connected • L: L-only connected to the bus • H: H-only connected • X: both connected

3.6.2 diag (Diagnostics query)

```
Command
diag
```

Run diagnostics of specified MX card.

Parameters

No parameters

Return value

No return value

```
Command
diag: res?
```

Read result of diagnostics.

Parameters

No parameters

Return value

<bus1><bus2>:<diag$_0$><diag$_1$>...<diag$_N$>

bus1, bus2	[enum]	State of L1/H1 (bus1) and L2/H2 (bus2): • *: bus is OK • S: bus is shorted
diag	[enum]	TP states: • *: TP is ok • L: Low-side relay of TP is permanently shorted • H: High-side relay of TP is permanently shorted • O: Any of low or high-side is permanently open

Command

```
diag:stat?{:<from>:<to>}
```

Read TP relays switch counters (switching statistics).

Parameters

from, to	[int]	Range of counters to read, zero-based. There is a limit of 20 counters, which can be read at once.

Return value

If argument from and to are not passed:
<count>:<date-time>
- count: total number of counters
- date-time: latest date and time when counters was saved

If passed, counters are returned:
<cnt$_0$>:<cnt$_1$>:...:<cnt$_N$>
- cnt: switch count, 24-bit number, 6-place hex format, up to 2^{24} - 1

3.6.3 route? (Routing state query)

Command

```
route?
```

Get state of routing of specified MX card. LCMX or HCMX card must be at selected position.

Parameters

No parameters

Return value

AL:<route>:CL:<route>:AH:<route>:CH:<route>

route	[string]	Routing configuration of specified card's bus (AL/CL...) to main bus 1 - 4. Always 4 characters: <bus1><bus2><bus3><bus4> ..where busN is equivalent position number (1 to 4) or "-" if not connected AL/CL can be connected only to main bus 1 - 4 L and AH/CH to bus 1 - 4 H. Example: --3- or -2-- or 1--4, ...

Example

AL: 1---: CL: --34: AH: 1---: CH: --34
AL/AH connected to Bus1-L/LH, CL/CH to Bus3 and 4-L/H.

```
Command
route?: m
```

Get state of routing of specified DEV card. DEV card is required on selected position.

Parameters

No parameters

Return value

MUX: B1L: <route>: B2L: <route>: ...: B3H: <route>: B4H: <route>: PS: <ps>: IMEAS: <imeas>

route	[string]	Routing configuration of specified main bus 1 - 4 L/H. Always 8 characters: 1234SMPQ ..where: • 1 to 4: CH1 to CH4 • S: SENS • M: MEAS • P: power-supply 1 • Q: power-supply 2 Character position is fixed. If any output is not connected, at specified position is "-".
ps	[string]	Power supply connection. 2 characters: <ps1><ps2> ..where psN is 1 or 2 at equivalent position or "-" if not connected.
imeas	[string]	Current measure activation. 1 character: • -: off • 1: active for power supply 1 • 2: active for power supply 2 (can be activated only for single power supply)

Example

MUX: B1L: -2------: B2L: --------: B3L: ------P-: B4L: --------: B1H: -2------:
B2H: --------: B3H: ------P-: B4H: --------: PS: --: IMEAS: -
Full configuration example.

3.7 probe (TP probe)

Probe commands.

TP probe work asynchronously. **The progress and results are sent via event channel.**

```
Header
f=probe
```

3.7.1 start (Start probe)

```
Command
start
```


Switch the probe ON.

Parameters

No parameters.

Return value

No return value

3.7.2 stop (Stop probe)

```
Command
stop
```

Switch the probe OFF.

Parameters

No parameters.

Return value

No return value

3.7.3 active? (Check probe status)

```
Command
active?
```

Get current probe status.

Parameters

No parameters.

Return value

`0` - disabled
`1` - active

3.7.4 Event channel

When active, the probe results are sent asynchronously via event channel.

```
Header
f=probe
```

Message data

```
Message
FI
```

Finding state. The probe is currently processing probe touch.

```
Message
NT
```

No-touch state. The probe finished processing without any touch. This will happen when the probe is removed

from the TP before the processing is done.

```
Message
<tp0>{:<tp1>:...:<tpN>}{...}
```

List of test-points found by the probe. The list is limited to maximum of 50 TP at once. If there is more than 50 TP detected, the result is stripped to 50 and the three points are appended at the end of list.

Parameters

`tp`	[int]	TP number
`...`	[string]	Indication that there is more than 50 TP in the list

3.8 mm (Multimeter)

Integrated multimeter commands.

```
Header
f=mm
```

3.8.1 route (Set routing)

```
Command
route:[{<mm-bus>:}<main-bus>]n-times
```

Clear and set the new multimeter's routing to main bus.

Note: this command will not set the routing switches (relays) immediately, it only configure the DMM to use them. Switches are connected at the time of measuring for necessary time only (until measuring is finished).

Parameters

`mm-bus`	[enum]	Multimeter's internal bus or line: • `l` = multimeter's LOW line • `h` = multiemter's HIGH line If this parameter is not passed, both buses (L and H) are selected by default.
`main-bus`	[enum]	Main bus: • `b1` = Bus 1 • `b2` = Bus 2 • `b3` = Bus 3 • `b4` = Bus 4 H/L of main bus is defined by mm-bus parameter. L of mm-bus is connect to L of main-bus (and the same for H).

Return value

No return value.

Examples

`route: b1`
Connect mutlimeter's both lines (L+H) to main bus 1.

`route: b2: l: b4: h: b3`

Connect mutlimeter's L line to main bus L 2 & 4 and H line to main bus H 2 & 3.

3.8.2 route? (Query routing)

```
Command
route?
```

Get currently configured multimeter's routing.

Parameters

No parameters

Return value

None: –
Configured: *[*{ <mm-bus>: } <main-bus>*]* *n-times*
(the combination {mm-bus:}main bus returns as many times as necessary to describe current configuration)

3.8.3 meas? (Perform a measurement)

```
Command
meas?: volt{: <range>}
```

Voltage measurement.

Parameters

range	[enum]	Voltage measurement range: • auto (auto-range, slow) • 2v5 • 25v • 250v

Return value

<result>: <selected-range>
- result: measured value, floating-point number, format "0.000", decimal point
- selected-range: selected range, typically result of auto-ranging

```
Command
meas?: res{: <range>}
```

Resistance measurement.

Parameters

range	[enum]	Resistance measurement range: • auto (auto-range, slow) • 1k • 100k • 10m

Return value

<result>: <resolution>: <v-meas>: <v-ref>
- result: measured value, floating-point number, format "0.0e+0", decimal point

- `resolution`: determined resolution of measured value, floating-point number, format "0.0e+0", decimal point
- `v-meas`: used measurement voltage, floating-point number, format "0.000000", decimal point
- `v-ref`: reference measurement voltage, floating-point number, format "0.000000", decimal point

The most important is result (the resistance value) and resolution (information about to correctly average the measurement value).

When open-loop (or larger resistance than 10MΩ), the result is `9.9e+37`.

```
Command
meas?:cont
```

Check continuity, very fast. It only compares with treshold level (about 500Ω).

Parameters

No parameters

Return value

- `9.9e+37`: open (represents infinite resistance)
- `0.0e+0`: short (zero resistance)

```
Command
meas?:cap
```

Capacitor measurement.

Parameters

No parameters

Return value

Capacitance value in exponential format (0.0e+0), always with decimal point.
If the value is above the possible maximum range, the return value is "9.9e+37".

```
Command
meas?:ind
```

Inductance measurement.

Parameters

No parameters

Return value

Inductance value in exponential format (0.0e+0), always with decimal point.
If the value is above the possible maximum range, the return value is "9.9e+37".

3.8.4 zero (Set zero)

```
Command
zero:volt
zero:res
zero:cap
zero:ind
```



```
zero:*rst
```

Set current measured value of voltage/resistance/capacity/inductance as offset for specific measurement.

Use `*rst` command to reset offsets of all measurements.

Parameters

No parameters

Return value

No return value

3.8.5 *rst (Reset)

```
Command
*rst
```

Reset all to default - routing and offsets.

Parameters

No parameters

Return value

No return value

3.9 fp (Front panel)

Front-panel LED control commands.

```
Header
f=fp
```

3.9.1 led (LED control)

```
Command
led{:<led-type>:<state>}N-times
```

Front-panel LED control. Diagnostics (dual-color R/G) and service (Y) LEDs on the front-panel are controlled by software. Other LEDs are controlled by the MX2400 system.

Parameters

`led-type`	[enum]	LED to control: • `diag-r`: diag LED, red • `diag-g`: diag LED, green • `service`: service LED, yellow • `*all`: all LEDs at once
`state`	[enum]	LED state: • `0`: off • `1` on

Return value

No return value

Example

led:*all:0
Turn off all LEDs

led:diag-g:1:service:1
Turn green diag LED and service LED

3.10 dio (DIO card control)

DIO card control commands.

```
Header
f=dio:a=<address>
```

a	[int]	Card address, zero-based. At specified address must be the DIO card.

3.10.1 in? (Read inputs)

```
Command
in?
```

Read current input states of DIO channels.

Parameters

No parameters

Return value

<in_0>:<in_1>:...:<in_N>
..where in_N is 8-bit number in 2-place hexadecimal format. LSB is lower input number.
There is as many in_N arguments as needed for number of inputs, i.e. for 64 IO card there will be 8 arguments.

3.10.2 out? (Read outputs)

```
Command
out?
```

Read current outputs states of DIO channels.

Parameters

No parameters

Return value

<out_0>:<out_1>:...:<out_N>
..where in_N is 8-bit number in 2-place hexadecimal format. LSB is lower input number.
There is as many out_N arguments as needed for number of inputs, i.e. for 64 IO card there will be 8 arguments.

3.10.3 out (Write outputs)

```
Command
out: <out0>: <out1>: ...: <outN>
```

Write output states of DIO channels. The states does not affect immediately, there is a trigger, which must be called to set states.

Parameters

`out`	[int]	Output values, 8-bit value, 2-place hex format. LSB = the lowest output number. There is as many out_N arguments as needed for number of outputs, i.e. for 64 IO card there will be 8 arguments.

Return value

No return value

3.10.4 clra (Clear all outputs)

```
Command
clra
```

Set all outputs of specified card to OFF.

Parameters

No parameters

Return value

No return value

3.10.5 trig (Output trigger)

```
Command
trig
```

Write the buffered states to DIO outputs. This will affect all DIO card at once (trigger broadcast).

Note: the "`a`" header is not used by the trigger, but the "dio" sub-system requires it - you can pass zero (0) here

Parameters

No parameters

Return value

No return value

3.10.6 count? (Channel count)

```
Command
count?
```

Count of DIO channels of specified card.

Parameters

No parameters

Return value

Number of DIO channels

3.10.7 cm (Set common config)

```
Command
cm: <group0>: <group1>: <group2>: <group3>
```

Common IN signal configuration for each input groups. The MXC2-DIO-64 card has 64 inputs, divided to 4 group by 16 inputs. Each group can be individually connected to specified common voltage.

Parameters

`group`	[enum]	Common signal configuration: • `0`: disconnected • `1`: GND • `2`: internal +5V • `3`: internal +24V

Return value

No return value

3.10.8 cm? (Query common config)

```
Command
cm?
```

Query for current common configuration.

Parameters

No parameters

Return value

<group0>: <group1>: <group2>: <group3>
The meaning is the same as set common config.

3.10.9 diag (Run diagnostics)

```
Command
diag
```

Run the internal card diagnostics.

Parameters

No parameters

Return value

No return value

3.10.1 diag? (Read diagnostics result)

```
Command
diag?
```

Read diagnostics results.

Parameters

No parameters

Return value

When not diagnosed: "ND"

Otherwise:
"<d_0><d_1><d_2>...<d_N>"
..where d is DIO_0 to DIO_N diagnostics state: "P" - PASS, "F" - FAIL
(number of letters equals to number of DIO channels)

3.10.1 Event channel

When detected any change on inputs of any DIO card, the notification is sent asynchronously via event channel.

```
Header
f=dio: a=<address>
```

address: address of DIO card, which inputs has changed

Message data

```
Message
in-changed
```

Change notification only. Input states must be then read by in? command.

3.11 bct (Batch-test)

Batch-test system control commands.

```
Header
f=bct
```

3.11.1 clr (Clear TP list)

```
Command
clr
```

Clear the list of loaded TPs.

Parameters

No parameters

Return value

No return value

3.11.2 load|cload|load? (Load TP list)

```
Command
load| cload{:g:<low>:<high>}N-times
load| cload{:g:l:<low0>:<low1>:...:<lowN>:h:<high0>:<high1>:...:<highN>}N-times
```

Load groups of TP configuration. The "cload" command will clear the list before loading, "load" command appends list.
These groups are automatically switched step-by-step by hardware after while batch-test is running in cooperation with multimeter device.

Capacity of TP list: 1024 (combined for L/H)
Example: capacity of 1024 TP with single connection always 1 TP to H and 1 TP to H the system can perform batch-test of 512 measurements, controlled by hardware

The TP number cannot be larger than 4095 (MX2400 currently supports up to 1152 TP, so at this time it does not matter).

Parameters

low	[int]	TP-low number(s)
high	[int]	TP-high number(s)

Return value

No return value

Example

`cload:g:1:2:g:10:16:g:20:30`
Clear the list and load 3 groups of TP with single TP, connected to low and high

`load:g:l:1:2:h:5:6:g:l:10:12:14:h:11:13:15`
Append to the TP list: 2 groups of TP with more TPs on low/high at one time

```
Command
load?
```

Returns number of loaded TPs.

Parameters

No parameters

Return value

Number of loaded TPs.

3.11.3 syst|syst? (Measurement system)

```
Command
syst:agt
```

Set "Agilent" as the measurement system. **This requires hardware connection between MX2400 and external Agilent multimeter** - using the SERVICE/SYNC connector of the master card, see the hardware manual.

In this mode the MX2400 triggers the external multimeter via dedicated signal and waits for completion via

another dedicated signal (using timeout). **The external multimeter must be properly configured** by user to measure the specified value using the external trigger before the batch-test will be started.

Parameters

No parameters

Return value

No return value

```
Command
syst:imm:<func>{:<range>}
```

Set "Internal multimeter" as the measurement system. The configuration (measurement type and range) is done immediately via this command.

Parameters

`func`	[enum]	Multimeter function: • `volt`: voltage • `res`: resistance • `cont`: continuity • `cap`: capacity • `ind`: inductance
`range`	[enum]	Range of specified function. The value depends on function used. See range descriptions in the meas? command of internal multimeter. **Default:** `auto` (if applicable, otherwise none)

Return value

No return value

```
Command
syst?
```

Currently configured system.

Parameters

No parameters

Return value

- `AGT` - Agilent system (external triggering)
- `IMM` - Internal multimeter

3.11.4 print? (Prints TP list)

```
Command
print?
```

Prints currently loaded TP list.

Parameters

No parameters

Return value

TP groups always in the format:
{ g: l: <low$_0$>: <low$_1$>: ...: <low$_N$>: h: <high$_0$>: <high$_1$>: ...: <high$_N$>}$^{N\text{-times}}$

3.11.5 res? (Measurement results)

```
Command
res?: <nr>
```

Read measurement results. This is applicable in case of use IMM measurement system (internal multimeter), which stores result directly inside the MX2400.

Parameters

nr	[int]	Index of result to read, zero-based. Maximum is 511 (1024 / 2 - 1). Valid count is determined by loaded groups of TP.

Return value

TP groups always in the format:
{ g: l: <low$_0$>: <low$_1$>: ...: <low$_N$>: h: <high$_0$>: <high$_1$>: ...: <high$_N$>}$^{N\text{-times}}$

3.11.6 run|run? (Run batch-test)

```
Command
run
```

Run the batch-test. Loaded TP groups in the list are required.

Parameters

No parameters

Return value

No return value

```
Command
run?
```

Query state of currently running batch-test.

Parameters

No parameters

Return value

State of batch-test:
- RUN: <current>: <cnt> - batch-test is running, indicating progress, current is currently processed TP group (zero based), cnt is number of TP groups
- DONE - finished
- TIMEOUT - external multimeter did not response in specified time (trigger response signal)
- ABORT - manually aborted before finish
- FAIL: <message> - batch-test failed, error message can be read

- – - none (no batch-test previously started)

3.11.7 abort (Abort batch-test)

```
Command
abort
```

Abort currently running batch-test.

Parameters

No parameters

Return value

No return value

