LACERTA M-Gen Stand-Alone AutoGuider's

USB interface protocol

of Firmware ver. 2.61

Table of contents

1. About the USB connection

The HandController / M-Gen device uses an USB-to-serial converter chip to get connection with the host / PC. This integrated circuit is made by FTDI Ltd. (Website: http://www.ftdichip.com)

Their appropriate D2XX driver must be installed on the PC. The latest driver can be downloaded from http://www.ftdichip.com/Drivers/D2XX.htm.

After installing it and connecting the M-Gen to the PC, the driver is loaded and will be ready to communicate after some seconds.

For example in case of using Windows, Your application must include the header file `ftd2xx.h` and use the supported library `ftd2xx.lib` . (Different libraries for 32 and 64 bit platforms are available in separate folders.) Serial communication is used then via the DLL functions. (See the programmer's guides at http://www.ftdichip.com/Support/Documents/ProgramGuides.htm)

As the M-Gen's interface is mainly based on serial communication, no exact implementations (code) will be shown how to use the driver calls, except some demonstration cases. In this document we use the C language functions for the examples and for the special D2XX-related functions.

The commands' name definitions header file is `mgen_usb.h` and may be found attached with this document.

All the functions of M-Gen documented here applies for the Firmware version indicated on the first page. As the Firmware has been developed continuously, they can differ or be non-implemented for the previous Fw. versions. I try to comment these information everywhere, where exactly known. These comments are placed in brackets and applies only for the Fw. versions that fulfill the condition. E.g. (≥2.10) means that the function / behaviour applies only from Fw. ver. 2.10 or later.

(Note that the Virtual COM Port (VCP) driver may also be used to communicate with the M-Gen, except the special ones.)

2. Communication bases

Two byte streams are present that are worked out by serial UART lines. The configuration of this is:

- 8 data bits
- 1 stop bit
- no parity bit
- (baud rate depends on the operational modes)

The M-Gen uses a simple concept about the communication. M-Gen is a “slave”, it never starts to send data to the PC, only answers for commands. The commands are binary (unsigned byte-based) and have a fixed structure, so are the answers. (“Fixed” does not mean that they can not be of variable length, there are format bits and length bytes to control these.)

We view the communication from the Master’s (PC) side. Commands and their parameters are *sent* to M-Gen and answers and content data are *received* from it.

In the protocol details later, a right-arrow (→) will indicate a command (sent bytes) and a left-arrow (←) an answer (received bytes). These should be interpreted as unsigned 8-bit integers. The symbol («) indicates the end of the command, no more bytes has to be sent or received.

2.1. BOOT and Application (state) modes

M-Gen's USB interface is accessible in it's both main state:

- BOOT mode (or called as 'Update mode'): there is a possibility to query actual versions, upload the firmwares, read / write the parameter (settings) memory;
- App. mode: the running firmware's commands are available.

These modes differ in communication and will be described along with the other operational modes, in the next section. We refer to these state modes as "BOOT" and "App.".

3. Operational modes

M-Gen communicates in two type of *operational modes*, of which the *Compatible Mode* is used after turn on. This is only for being compatible and identifiable for other existing devices using the same USB-to-serial chip (e.g. Ursa Minor devices). To access the functionalities Your application must enter the *Normal Mode*.

Cases when M-Gen switches to *Compatible Mode*:

- after starting the firmware (or power-on or external Reset);
- switching to BOOT mode;
- USB power is lost (e.g. plugged out). (≥2.04)

The baud rate of this mode is **9600**.

Case when M-Gen switches to *Normal Mode*:

- a command received in *Compatible Mode* to do so.

The baud rate of this mode is **250000**.

Both *operational modes* is present for both state modes BOOT and App., resulting in four different exact communication behaviour. Since *Compatible mode* is useless in controlling the M-Gen, I will describe only *Normal mode* in details.

The current mode of M-Gen can be detected anytime, I will present an example procedure how to do it.

3.1. *Compatible mode* protocol

The M-Gen by default starts working in this mode (either BOOT or App.). Baud rate must be set to 9600 and the following commands are valid:

► MGCP_QUERY_DEVICE (`0xaa`)

`0xaa` →	(1st byte is always the command, not indicated later)
`0x01` →	length of parameters (= number of following bytes), ≥1
`0x01` →	“query device ID” sub-command
← `0x55`	(means an incoming “compatible answer”)
← `0x03`	length

If M-Gen is in BOOT mode, it answers:

← `0x01 0x80 0x01` «

Or else, M-Gen is in App. mode and answers:

← `0x01 0x80 0x02` «

If the 3rd sent byte is not `0x01`, the device will not answer any bytes back, but will read (require) the given number of bytes. The same applies if the 2nd byte (length) is zero.

► MGCP_ENTER_NORMAL_MODE (`0x42`)

`0x42` → «

No more bytes are read and no bytes are answered, the device immediately switches to *Normal mode* with the increased baud rate of 250000. Wait at least 10 ms before sending new command to the M-Gen. (Include the driver’s / USB delay, so in practice, it is advised to wait about 100 ms and set minimal transmit delay by D2XX driver (2 ms).)

3.2. *Normal mode* protocol

If switched to this mode yet, Your application can use the main M-Gen protocol. The baud rate is 250000.

The BOOT and App. mode command list is very different, they are introduced separately in the next subsections.

M-Gen will be waiting for the next command byte as being idle. The basis of executing a command is the following:

- PC sends the command byte to M-Gen
- if M-Gen's firmware does not know that command, no answer will come. The timeout value is 1 second.
- if M-Gen knows that command, it will answer an appropriate (usually the same) byte as an ACKnowledgment.
- PC sends and/or receives bytes according to the command's structure

Each command has its own structure and even an own timeout value. These will be represented at the commands' details.

After a command has been completed, M-Gen is waiting for the next one. Hence, PC and M-Gen must be in sync for the entire communication time.

4. BOOT mode commands

Note that M-Gen does not respond to an unknown command byte and waits for the next byte as the new command.

► MGCMD_NOP0 (`0x00`)
► MGCMD_NOP1 (`0xff`)

"No Operation" commands. The ACK byte is inverted, indicating M-Gen is currently in BOOT mode.

```
0x00 →
    ← 0xff «      inverted ACK for NOP
```

or

```
0xff →
    ← 0x00 «      ...as well.
```

► MGCMDB_GET_VERSION (`0x14`)

Gets the boot software's version number. This tells the PC what exact hardware the M-Gen's HandController has. (There are some tiny differences.)

```
0x14 →
    ← 0x14            ACK.
    ← BOOTVER «       The desired boot software version byte.
```

The valid version bytes and their meaning:

`0x12,0x13`	M-Gen's hardware is the *old* one.
`0x14,0x16`	M-Gen's hardware is the *new* one.

► MGCMDB_GET_FW_VERSION (`0x2d`)

Gets the uploaded Firmware's version number if there is any. The version number is a 16-bit integer and must be interpreted / displayed as hexadecimal. For example `0x0212` stands for Fw. ver. 2.12.

```
0x2d →
    ← 0x2d        ACK.
```

If the firmware is valid:

← `0x00`	No error.
← `FW_LO FW_HI` «	Firmware version low and high byte. (version = 256*FW_HI + FW_LO)

If the firmware is invalid (missing or damaged):

← `0xce` «	CRC (checksum) error!

► MGCMDB_GET_CAMERA_VERSIONS (`0x2e`)

Gets the boot and uploaded Firmware's version of the Camera if there is any. The Fw. version number is a 16-bit integer and must be interpreted / displayed as hexadecimal. For example `0x0212` stands for Fw. ver. 2.12.

The Camera must be connected to the HandController. If the query fails, DC power supply should be applied for the HandController and retry the command.

```
0x2e →
    ← 0x2e        ACK.
```

First, the Camera boot version is answered.
If an error occured:

← `ERROR` « Error code. Valid values are:

`0x2e`	Can't start the Camera (for any reason)
`0xc0,0xc1`	Camera not connected properly
`0xa7`	DC power voltage missing / wrong (if needed)
`0xcf`	Communication error (at any time)
`0xce`	Camera reported an error while querying
?	(Other bytes might be rarely read at special cases or when Camera is damaged, even 0.)

On success:

← `0x00`	Ok.
← `CAMBOOT` «	Camera's boot version byte.

The valid version bytes and their meaning:

`0x11`	M-Gen Camera's hardware is the *old* one.
`0x14`	M-Gen Camera's hardware is the *new* one.

Second, the Camera's firmware version is answered.
If an error occured:

← `ERROR` « Error code. Valid values are:

`0xcf`	Communication error (at any time)
`0xce`	Camera reported an error while querying
`0xc5`	CRC error. Firmware is invalid on it

On success:

← `0x00`	Ok.
← `FW_LO FW_HI` «	Camera's Firmware version low and high byte. (version = 256*FW_HI + FW_LO)

(Note that if You get a suspicious version value (e.g. `0xd5c2`), the Camera may be working improperly.)

► MGCMDB_RUN_FIRMWARE (`0xe1`)

Tries to starts the uploaded Firmware of the M-Gen. It does a CRC check before as at each startup.

`0xe1` → «

This command does not answer with an ACK!

After applying this, M-Gen gets back into *Compatible mode.* Use that protocol from now.

If CRC error is detected, the M-Gen restarts in BOOT mode (while displaying "NO FIRMWARE" on the LCD).

If the Firmware is valid, M-Gen continues working in App. Mode.

Wait at least 1 second before applying another command.

To check the success of the command use MGCP_QUERY_DEVICE command to detect whether M-Gen has got into BOOT or App. mode.

► MGCMDB_POWER_OFF (`0xe2`)

This command immediately makes the M-Gen go to power-down state. The four red LEDs flash just as when a power source is applied first to it. Further communication is impossible until the device is turned on.

`0xe2` → «

This command does not answer with an ACK!

5. Application mode commands

As in BOOT mode, M-Gen does not respond to an unknown command. There may be undocumented commands included in the Firmware, please avoid executing these to keep the device('s state) consistent and workable.

► MGCMD_NOP0 (`0x00`)
► MGCMD_NOP1 (`0xff`)

"No Operation" commands. The ACK byte is not inverted, indicating M-Gen is currently in App. mode.

```
0x00 →
    ← 0x00 «      ACK
```
or
```
0xff →
    ← 0xff «      ...as well.
```

► MGCMD_ENTER_BOOT_MODE (`0xeb`)

After applying this the device will immediately restart in BOOT mode.

```
0xeb →  «
```

M-Gen continues communication is *Compatible mode*.
Wait at least 1 second before applying a new command.

► MGCMD_GET_FW_VERSION (`0x03`)

Gets the running Firmware's version number. The version number is a 16-bit integer and must be interpreted / displayed as hexadecimal. For example `0x0212` stands for Fw. ver. 2.12.

```
0x03 →
    ← 0x03              ACK.
    ← FW_LO FW_HI «     Firmware version low and high byte.
                        (version = 256*FW_HI + FW_LO)
```

► MGCMD_READ_ADCS (`0xa0`)

Gets the latest 10-bit ADC (Analog-to-Digital Converter) conversion values. Five analog signal (index 0 to 4) is connected to the microcontroller unit, the (approximated) voltage can be derived from the result as seen below.

(This is a development command, the result might very rarely be wrong (the lowest 2 bits of 10 only) due to the lack of mutual exclusion. If You want to read a precise and trusty value, read ADCs more times and take the medians.)

`0xa0` →	
← `0xa0`	ACK.
(5x) ← `ADC_LO ADC_HI` «	The 16-bit (unsigned) integer values of the ADCs. The bits are left aligned, the lowest 6 bits are always zero.

Converting values to voltage (in Volt unit):

At index 0, the logic supply voltage: $V_{VCC} = 1.6813e\text{-}4 * ADC[0]$
At index 1, the DC input voltage: $V_{IN} = 3.1364e\text{-}4 * ADC[1]$
(At index 2 and 3, special voltages, used only by the old type of M-Gen hardware. We skip these.)
At index 4, an internal reference voltage: $V_{bg} = 3.91e\text{-}5 * ADC[4]$

Notes on the values:

If ADCs are working properly, V_{bg} must be somewhere around 1.23 V. More than 10% difference must be showing some kind of internal problem.

The typical measured logic supply V_{VCC} is 4.8 - 5.1 V when operating from DC power. If USB supply is used V_{VCC} can be lower, about 4.7 V. The logic works stable over 4.5 Volts.

The DC input must be between 9 and 15 Volts. Up to 20.5 V can be measured but it's not recommended to use as the input voltage. (The absolute maximum tolerable peak voltage is 20 V). Note that the polarity protector diode lowers the measured DC voltage by 0.15 - 0.3 V.

The ADC's and circuit's overall conversion error is less than 10%.

► MGCMD_GET_LAST_FRAME (`0x9d`)

The last guiding frame's data can be read that has been got from the Camera.

`0x9d` →	
← `0x9d`	ACK.
`FLAGS` →	Command's flags (what to query).

If FLAGS > 1, an error indicator returns and no more bytes come:

← `0xff` «	Means that FLAGS has invalid value.

else

← `FR_NUM`	The frame index and 'star present' flag. The frame index is 'FR_NUM modulo 64' and is increasing as new frames come from the Camera. Bit 6 of FR_NUM indicates when a star was present on this frame (1) or not (0). Bit 7 is zero. (Use the index to detect if a new frame is present, apply this command periodically.)

If bit 0 of FLAGS is 1, the drift values are sent too:

← `RA_FRAC RA_INT`	RA drift, a 8+8 bit signed fixed-point value. The unit is pixels (CCD horizontal pixel size). Real value is calculated from unsigned bytes as: D_{RA} = (RA_FRAC + RA_INT*256) / 256.0 If (RA_INT > 127) $D_{RA} = D_{RA} - 256$
← `DEC_FRAC DEC_INT` «	The same as above, but for the DEC drift.

If the device has calibration data, RA and DEC drifts are calculated using those vectors as a coordinate system. If there is no calibration data, RA and DEC are the X and Y coordinates, so X and Y drifts are sent back on reply of this command.

► MGCMD_IO_FUNCTIONS (0x5d)

A command that groups several input/output functions. Using these Your application is able to implement a virtual user interface for the M-Gen.

`0x5d →`	
`← 0x5d`	ACK.
`SUBFUNC →`	Command’s sub function identifier. The protocols are unique to the sub functions.

If an invalid SUBFUNC value is sent, the command ends («) and M-Gen is waiting for the next command byte. (For example if MGIO_GET_LED_STATES times out but others are working, it means that the firmware ver. is below 2.12.)

Values for SUBFUNC:

➢ MGIO_INSERT_BUTTON (0x01)

`CODE →` «	The button’s code to insert into the button input buffer. (It acts as a ‘keydown/keyup’ event in the UI of M-Gen.)

For ‘keydown’ event, CODE’s MSB (bit 7) is zero.
For ‘keyup’ event, CODE’s MSB is one. (≥2.10)

The lower 3 bits of CODE identifies the button, all the other bits must be zero!

0	ESC
1	SET
2	LEFT
3	RIGHT
4	UP
5	DOWN
6	“LONG ESC” (only a ‘keydown’ event makes sense) (≥2.02)

➢ MGIO_GET_LED_STATES (0x0a) (≥2.12)

`FLAGS →`	Flags telling what to query about the LED indicators.

If bit 0 of FLAGS is one, the current states of the LEDs are answered.

`← LEDS` «	LED states (on(1) / off(0)) encoded in the bits:

bit 0	blue LED: “exposure focus line active”
bit 1	green LED: “exposure shutter line active”

bit 2	red LED (up): “DEC- correction active”
bit 3	red LED (down): “DEC+ correction active”
bit 4	red LED (left): “RA- correction active”
bit 5	red LED (right): “RA+ correction active”

➢ MGIO_READ_DISPLAY `(0x0d)`

With this IO function You can read the display buffer content of the M-Gen.

`ADDR_L ADDR_H` → The address to start read from, low and high bytes. The lowest 10 bits are used only.

A display byte is 8 display bits shaping a column. LSB is at the top.
M-Gen’s display (128x64 bits) bytes are addressed this way:

0	1	...	126	127
128	129	...	254	255
⋮				⋮
896	897	...	1022	1023

A bit value of one indicates the LCD pixel lit.

`COUNT` → Number of display bytes to read sequentially (increasing addr.). (Zero count is valid and no data bytes will be sent in this case.)

← `D0 D1 ...` « COUNT pieces of bytes answered as the content data.

Since 256 bytes can not be read in one command, it is advised to read the display content with a block size of 128 bytes, which slices into 8 rows of the LCD. 255 bytes can be read, so the entire display content can be read with 5 display read commands (4 x 255 + 4 bytes) as the fastest way.

Note that the display buffer is not the same as the exact bits displayed on the LCD. The LCD’s content can not read back (only refreshed) and there may be drawing operations in progress on the display buffer while reading its content. (The USB/communication process works asynchronously from the others.) This can lead to “flashing” screen elements, but nevermind it.

► MGCMD_RD_FUNCTIONS (`0xa8`) (≥2.04)

A command that groups functions for the Random Displacement (or dithering).

`0xa8` →
  ← `0xa8` ACK.

`FLAGS` → Command's flags.

Flags of bit 2 to 7 must be zero, or else M-Gen says 'error':

← `0xff` « Wrong flags given.
No more bytes are coming.

Bugfix: works for ≥2.30 only. Previous versions return `0x03`.
Workaround is avoiding sending invalid command flags.

If bits 1:0 are
`00` queries dithering (RD's) state.
← `STATE` « The internal state byte. Meanings of it:

bit 0,1	Always one.
bit 2,3	<dontcare>
bit 4	One if RD is active (moving the mount, measuring new position).
bit 5	<internal, dontcare>
bit 6	One if RD is enabled (at the next trigger).
bit 7	'RD enabled' state at the last trigger.

If bits 1:0 are
`01` forces to start a dithering (generates the next guiding position).

If no error
← `0x00` «

If dithering is already active
← `STATE` « The same non-zero state byte as for bits `00` above.

If bits 1:0 are
`10` all RD variables are read:

← `COUNT` Number of bytes to be sent by M-Gen. (3 in fw. 2.30)
← `STATE` The same as for bits `00` above, except that the LSB indicate the RD's mode:
0 Uniform square

1	Square snake
← `SIZE_L SIZE_H`	RD size. This fractional number is stored as "decimal", calculate as: RD_size = SIZE_H + SIZE_L / 100.0

(« Bytes end when COUNT number of bytes has been reached. If COUNT is 1, only STATE is sent.)

If bits 1:0 are
`11` all RD variables could be set:

← `COUNT`	Number of bytes M-Gen wants to receive. (3 in fw. 2.30)
`FLAGS` →	Sets new RD mode or enables/disables it. If bit 4 of FLAGS is set, bits 1:0 are copied to RD's mode. (Though bit 1 is dontcare for fw 2.30.) If bit 5 of FLAGS is set, bit 6 sets RD's new enabled state. (1=enabled, 0=disabled)
`SIZE_L SIZE_H` →	New RD size value in the same 'decimal' format as Above. If zero, RD size does not change.
← `0x00`	This tells OK. No more bytes are coming.

► MGCMD_EXPO_FUNCTIONS (`0xa2`) (≥2.11)

A command that groups functions related to exposure control. Only one function is available currently.

`0xa2` →		
	← `0xa2`	ACK.
`SUBFUNC` →		Sub-function identifier and possibly flags.

Values for SUBFUNC:

➢ MGEXP_SET_EXTERNAL (`0x0?`)

Tells M-Gen the external exposure state. Bit 1 of this value gives the new 'external exposure' state.

If the lowest bit is not one (zero is reserved), an error is reported:
← `0xe0` « "Unknown external expo function"

Otherwise,
← `0x00` « Ok.

If an 'External exposure' state is set (one), M-Gen will save guiding drifts into the open file. This is used to save the data if an external exposure controller is used. M-Gen saves data by default only if the internal AutoExposure is used for exposure control.

You can use the predefined SUBFUNC values:

MGEXP_SET_EXTERNAL_OFF	to tell exposure is off,
MGEXP_SET_EXTERNAL_ON	to tell exposure is on.

Other (masked `0xf0`) values for SUBFUNC are reserved and give error:
← `0xe0` « "Unknown expo function"

► MGCMD_AG_FUNCTIONS (`0xca`) (≥2.60)

A command that groups functions related to control autoguiding and related procedures.

`0xca` →
  ← `0xca` ACK.

`SUBFUNC` → Sub-function identifier and possibly flags.

In the case of an invalid SUBFUNC value:
  ← `0xe0` « “Unknown autoguiding function”. Communication ends.

Values for SUBFUNC and the following protocol:

➢ MGAG_START (`0x03`) (≥2.60)

Tells M-Gen to start the autoguiding with the current star.

If AG can not be started (for some reason):
  ← `0x01` « “Can’t start autoguiding, no star is seen”
  or
  ← `0x02` « “Can’t start autoguiding, invalid screen is active”

In the case of locked UI loop in the software, when USB process can’t start any else function:
  ← `0xf0` « “Can’t do functions”

Otherwise,
  ← `0x00` « Ok, autoguiding has been enabled.

The functionality is equal to pressing the ‘AG start’ button on the guiding screen. Note that starting the autoguiding when it is already active changes the guiding center position to the latest available star position as AG was started just that time.

➢ MGAG_STOP (0x01) (≥2.60)

Tells M-Gen to stop the autoguiding.

In the case of locked UI loop in the software, when USB process can't start any else function:

← `0xf0` «	"Can't do functions"

Otherwise,

← `0x00` «	Ok.

The functionality is equal to pressing the 'AG stop' button on the guiding screen. Nothing happens (but 'Ok' is returned) if there was no autoguiding active.

➢ MGAG_QUERY (0x10) (≥2.60)

To receive information about the current autoguiding.

In the case of locked UI loop in the software, when USB process can't access AG data:

← `0xf0` «	"Can't access data"

Otherwise,

← `0x00` «	Ok.
`flags` →	Flags for the required data to be read.

Bits for flags (one/high means data is required):

bit 0	ignored
bit 1	to get Autoguiding process' state
bit 2	to get info about the current / latest frame
bit 3-7	ignored

Autoguiding state byte is sent to the PC if bit 1 of `flags` is high:

← `AG_state`

`0x00`	AG is not active.
`0x01`	AG is active.

Info on the current / latest frame is the following:

← `frame_info`	Frame index and 'star present' flag
← `pos_X (3 bytes)`	Raw CCD X coordinate of the last seen star
← `pos_Y (3 bytes)`	Raw CCD Y coordinate…
← `d_RA (2 bytes)`	Drift in RA (in pixels), if AG is active.

← `d_DEC (2 bytes)`	Drift in DEC…
← `peak`	Peak value of the star (used) pixels

Frame index is an increasing number as new CCD frames are read. The lower 6 bits are provided. If the Camera can see and evaluate a star in it, bit 6 is set (otherwise it's zero). Bit 7 is always zero.
Raw CCD coordinates show where the last star has been measured. (If the 'star present' flag is zero, this value is the lastest one when that flag was one.) It's a signed 16.8 bit fixed point number. (The lower 8 bits are the fractional part, the next 15 bits is the integer part and bit 23 is the sign.). (Note that the CCD pixels are not exactly square, 4.85 um horizontal (X) and 4.65 um vertical (Y)! For binning modes, these sizes are multiplied.)
Drift is the same value as the MGen uses for autoguiding and display. It's only valid if AG is active. It holds the latest measured value as is for the raw coordinates. The value is a signed 8.8 bit fixed point number. (Note that the drift values are transformed and corrected so that the CCD had 4.85 x 4.85 um square pixels (same for binned).)

➢ MGAG_START_CALIBRATION (0x20) (≥2.60)

The commands starts a calibration procedure if available. The command is non-blocking, MGen will answer this command and will be running the process, while new commands may be issued.

Cases and return values when the procedure can't be started: (Command ends.)

← 0xf2 «	"Camera is off"
← 0xf3 «	"AutoGuiding is enabled / active"
← 0xf1 «	"An other command is already under execution"
← 0xf0 «	"UI is locked, can't execute command" (on some screens)

Otherwise,
← 0x00 « Ok, calibration has started.

The functionality is equal to pressing the 'Calibrate' button on the guiding screen. A running calibration can be canceled by pressing ESC on the HandController, even it was issued from USB.

➢ MGAG_QUERY_CALIBRATION (0x29) (≥2.60)

To read the current status of the calibration procedure and the return value (success' state) of the last run.

In the case of locked UI loop in the software, when USB process can't start any else function:
← 0xf0 « "Can't do functions". (Comm. ends.)

Otherwise,
← 0x00 « Ok.

← calib_state

0x00	Not yet started. (First state after power-on.)
0x01	Measuring start position.
0x02	Moving DEC, eliminating backlash.
0x03	Measuring / moving DEC.
0x04	Measuring / moving RA.
0x05	Almost done, moving DEC back to original pos.
0xff	A calibration has ended.

← calib_error

When calib_state is not 0xff, calib_error's value is invalid.

Else, `calib_error` means the following:

`0x00`	No error, success.
`0x01`	The user has canceled the calibration.
`0x02`	Star has been lost (or wasn't present).
`0x04`	Fatal position error detected.
`0x05`	Orientation error detected.

➢ MGAG_CANCEL_CALIBRATION (0x2c) (≥2.60)

Cancels the currently running calibration, just as ESC was pressed on the HandController.

Cases and return values when the procedure can't be canceled: (Command ends.)

← `0xf1` «	"An other command is already under execution"
← `0xf0` «	"UI is locked, can't execute command" (on some screens)

Otherwise,
← `0x00` « Ok.

The execution of caceling command is somewhat later than the command has sent the return value 'ok'. The user must use the state query command (`0x29`) to ensure that the procedure has really ended.

➢ MGAG_START_STARSEARCH (`0x30`) (≥2.60)

Starts a new star search procedure if possible.

Cases and return values when the procedure can't be started: (Command ends.)

← `0xf2` «	"Camera is off"
← `0xf3` «	"AutoGuiding is enabled / active"
← `0xf1` «	"An other command is already under execution"
← `0xf0` «	"UI is locked, can't execute command" (on some screens)

Otherwise,

← `0x00` «	Ok, a new star search will be executed.

Then the parameters required for star search are:

`gain` →	Gain value for the Camera. (2 to 9)
`expo_ms (2 bytes)` →	Exposure time used. (50 to 4000) LSB first.

← `num_stars`	The number of found stars are sent back as an answer.

The execution of star search command is blocking! The HandController won't answer until the procedure has ended. To be able to do this the user must set the read timeout of the FTDI D2XX driver enough high: `expo_ms` milliseconds plus the readout and frame processing (max. 10 seconds). Using 15 seconds timeout must work in all cases. (A typical run time is the exposure time + 2 seconds.)

➢ MGAG_GET_STAR_DATA (`0x39`) (≥2.60)

Reads the data for a lately found star. This command must be used after a successful star search, when `num_stars` was non-zero.

Cases and return values when the procedure can't be started: (Command ends.)

← `0xf2` «	"Camera is off"
← `0xf3` «	"AutoGuiding is enabled / active"
← `0xf1` «	"An other command is already under execution"
← `0xf0` «	"UI is locked, can't execute command" (on some screens)

Otherwise,

← `0x00` «	Ok, a new star search will be executed.

`star_idx` →	The index of the star to get info for. (0 to `num_stars-1`)

← `star_idx`	On success, the star index is echoed back.
or	
← `0xff`	Can't get info for this star. (Wrong index etc.) Comm. ends.

If succeeds, the info is sent back for that star:

← `pos_X (2 bytes)`	X coordinate of the star on the CCD.
← `pos_Y (2 bytes)`	Y coord.
← `bright (2 bytes)`	'Brightness' of the star. (15 bits!)
← `pixels`	Used pixels ('area') of the star's image.
← `peak`	Peak value of the star (8-bit absolute)

The `bright` value's highest bit indicates if the star had a saturated pixel (1). The lower 15 bits allows the max. value of 32767, brighter stars should be viewed with much lower gain/exposure time.

The saturated pixel's (peak) value can be lower than 255. This is due to the interlaced readout CCD, the second field has to be scaled down to match the first field's shorter exposure time.

The found stars are sorted by brightness. Always the brightest star can be found at index 0, which is possibly the best option for guiding. Note that the star search function is mostly insensitive to hot pixels but is sensitive to 'hot pixel structures'. On this rare case the user must remember the position of these structures (±1 pixels) and ignore them on the stars' list. (They may not have saturated pixels! Use the position only for detection.)

➢ MGAG_SET_GUIWIN `(0x3f)` (≥2.60)

Use this command to set a new guiding (window) position. After, the star located in the guiding window will be followed by the camera and can be used for guiding.

Cases and return values when the procedure can't be started: (Command ends.)

← `0xf2` «	"Camera is off"
← `0xf3` «	"AutoGuiding is enabled / active"
← `0xf1` «	"An other command is already under execution"
← `0xf0` «	"UI is locked, can't execute command" (on some screens)

Otherwise,

← `0x00` «	Ok, a new position will be set.

`pos_X (2 bytes)` →	The CCD X position of the star to be followed.
`pos_Y (2 bytes)` →	Y position.

Use the star positions got from the get-star-data (0x39) command.

If a position value is set to negative, that coordinate will not be changed.

➢ MGAG_SET_IMAGING `(0x91)` (≥2.60)

This command changes the imaging parameters like gain, exposure time and threshold. Gain and threshold is changed immediately (the current exposure will be read out using these parameters), while the new exposure time will take effect at taking the next frame. If the user wants to get valid data (position, peak etc.) with the new imaging variables, the user must query the current frame number right after changing the img. Parameters, then wait for the next frame to be ready, that frame will be valid. (Or else the user could force to restart the exposures immediately by applying MGAG_SET_GUIWIN function with negative valued parameters.)
It's allowed to change the parameters while autoguiding is active but keep in mind the effects above.

Cases and return values when the procedure can't be started: (Command ends.)

← `0xf2` «	"Camera is off"
← `0xf1` «	"An other command is already under execution"
← `0xf0` «	"UI is locked, can't execute command" (on some screens)

Otherwise,

← `0x00` «	Ok.

`gain` →	New gain value (2 to 9)
`expo_ms (2 bytes)` →	Exposure time in ms. (50 to 4000)
`threshold` →	New threshold value (1 to 99)

➢ MGAG_GET_IMAGING `(0x92)` (≥2.61)

This command reads the current imaging parameters like gain, exposure time and threshold.

Cases and return values when the procedure can't be started: (Command ends.)

← `0xf1` «	"An other command is already under execution"
← `0xf0` «	"UI is locked, can't execute command" (on some screens)

Otherwise,

← `0x00` «	Ok.

←	`gain`	Current gain value
←	`expo_ms (2 bytes)`	Current exposure time in ms.
←	`threshold`	Current threshold value

- ➢ MGAG_CAMERA_ON / …_OFF `(0xc1 / 0xc0)` (≥2.60)

Turns on or off the Camera. Turn-on needs more time to complete, communication timeout must be set to at least 3 seconds.
Note that turning on the Camera won't set the imaging parameters so the user must apply MGAG_SET_IMAGING and MGAG_SET_GUIWIN subcommands after turn-on to make sure that the Camera will try to follow a star.

Cases and return values when the procedure can't be started: (Command ends.)

← `0xf1` «		"An other command is already under execution"
← `0xf0` «		"UI is locked, can't execute command" (on some screens)

Otherwise,

← `0x00` «	Ok.

6. Examples

I try to make code implementation easier with introducing some C/C++ code examples, which may seem to be complicated. Feel free to restructure them.

The following conventions and functions are used:

//	comment is shown with *another font and color*
`BYTE`	is defined type for `unsigned char` (or `int8_t`)

`SetBaud(int N)`	Sets the serial baud rate to N.
`Wait_ms(int N)`	Waits N milliseconds.
`FlushIn()`	Flushes (clears) the read buffer.
`SendToMGen(int N, ...)`	This function sends N bytes to M-Gen. (→)
`ReadFromMGen(int N, BYTE *var)`	This function reads N bytes from M-Gen (←) into array `var`. Returns `N` on success, or `-1` if not enough byte got before the timeout.

6.1. Connecting to M-Gen at random time

First of all, the data channel to an appropriate FTDI device must be opened. (Both D2XX or VCP.) If this fails (no device can be found), M-Gen may be used by another program.

The input variable `BOOT` tells what mode You want to connect to M-Gen: it is zero if App. mode is required. Since the *Compatible* mode is useless for doing functions, we connect always into *Normal* mode.

Read timeout is set to 1 seconds. (The 'latency timer' of FTDI driver is set to the minimal 2 ms.)

```
int ConnectToMGen(int BOOT)
{
    BYTE cid1[5] = {0x55,0x03,0x01,0x80,0x01};
    BYTE cid2[5] = {0x55,0x03,0x01,0x80,0x02};
    BYTE readid[5], nop_answer;

    SetBaud(9600);
    FlushIn();
    // We check if MGen is in Compatible mode
    SendToMGen(3, 0xaa, 0x01, 0x01);
    if (5 == ReadFromMGen(5, readid))
    {
        if (memcmp(readid, cid1) && memcmp(readid, cid2))
            return 1;

        // we are in Compatible mode... let's switch to Normal
        SendToMGen(1, 0x42);
        Wait_ms(100);
    }

    // (try to) communicate in Normal mode
    SetBaud(250000);

    SendToMGen(1, 0x00);
    if (ReadFromMGen(1, &nop_answer) != 1) return 1;

    if (nop_answer != (BOOT ? 0xff : 0x00))
    {
        // another state mode detected...let's change it
        SendToMGen(BOOT ? 0xeb : 0xe1);
        Wait_ms(100);
        FlushIn();
        Wait_ms(1000);
        SetBaud(9600);

        // MGen must be in Compatible mode now
        SendToMGen(3, 0xaa, 0x01, 0x01);
        if ((ReadFromMGen(5, readid) != 5) ||
             (memcmp(readid, cid1) && memcmp(readid, cid2))
            )
            return 1;

        // switch to Normal
        SendToMGen(0x42);
        Wait_ms(200);
        SetBaud(250000);
    }
    // the required mode is active... Done.
    return 0;
}
```

Be careful using the code above, because it could instantly go to BOOT mode if that was required even if M-Gen was autoguiding or having an open file. (The file will be broken, some data may not be written yet.)

The procedure does not work properly if the serial communication has gone out of sync before.

6.2. Special D2XX features

Using the D2XX library allows two extra features for the M-Gen with new hardware and only one for the old hardware. New HW is indicated by a BOOT version ≥0x14 or can be checked visually in BOOT mode, a small star is shown next to the text 'UPDATE MODE'.

Using the VCP driver these features are inaccessible.

The examples below uses direct calls to the D2XX API. See the Programmer's Guide for more details (1).

6.2.1. Hardware reset

CBUS bit 0 drives the reset pin of the MicroController Unit inside the M-Gen. If this is configured as 'output high', MCU will be under a reset condition. Other configurations will not activate the reset line.

The next function may be used to trigger a HW reset. (`cbus_dir` and `cbus_val` variables hold the current state of the CBUS pins, `hDev` holds the opened device handle.)

```
UCHAR cbus_dir, cbus_val;

int Reset_MGen(FT_HANDLE hDev)
{
   cbus_dir |= (1<<0);      // bit 0 output
   cbus_val |= (1<<0);      // bit 0 high
   UCHAR b = (cbus_dir << 4) + cbus_val;
   FT_STATUS ftS = FT_SetBitMode(hDev, b, 0x20);
   if (FT_OK != ftS) return 1;

   Wait_ms(50);

   cbus_val &= ~(1<<0);     // bit 0 low
   b = (cbus_dir << 4) + cbus_val;
   ftS = FT_SetBitMode(hDev, b, 0x20);
   return (FT_OK != ftS);
}
```

Hardware reset should never done to M-Gen in normal use! Doing such kind of external reset will probably put M-Gen into power-save (turn off) mode.

6.2.2. Turning on

This feature applies only for new hardware M-Gens.

CBUS bit 1 drives the ESC button's line directly. If this is configured as 'output high', the ESC will be held 'pressed'. This is a way to remotely turn-on M-Gen.

A function to 'press ESC from software' for a `time_ms` period is like:

```
UCHAR cbus_dir, cbus_val;

int PressESC_MGen(FT_HANDLE hDev, int time_ms)
{
    cbus_dir |= (1<<1);      // bit 1 output
    cbus_val |= (1<<1);      // bit 1 high
    UCHAR b = (cbus_dir << 4) + cbus_val;
    FT_STATUS ftS = FT_SetBitMode(hDev, b, 0x20);
    if (FT_OK != ftS) return 1;

    Wait_ms(time_ms);

    cbus_val &= ~(1<<1);     // bit 1 low
    b = (cbus_dir << 4) + cbus_val;
    ftS = FT_SetBitMode(hDev, b, 0x20);
    return (FT_OK != ftS);
}
```

6.3. Communication examples

6.3.1. Calibration

If the device is in the proper state, calibration can be started with the following sequence:

```
0xca →         'AutoGuiding function'
   ← 0xca         Ok.
0x20 →         'Start calibration'
   ← 0x00         Ok, started.  (Error code is here when can't be started.)
```

Now the user should poll for the state of the calibration. For example:

```
0xca →         'AutoGuiding function'
   ← 0xca         Ok.
0x29 →         'Query calibration state'
   ← 0x00         Ok.
   ← 0x03         Calibration state is: DEC movements being done
   ← 0x00         (don't care)
```

Until the end when he reads the 'ended' state:

```
0xca →         'AutoGuiding function'
   ← 0xca         Ok.
0x29 →         'Query calibration state'
   ← 0x00         Ok.
   ← 0xff         Calibration has ended.
   ← 0x00         Result is: success
```

6.3.2. Star search and selection

If the device is in the proper state, star search can be executed as follows:

```
0xca →         'AutoGuiding function'
   ← 0xca         Ok.
0x30 →         'Star search'
   ← 0x00         Ok, allowed. (Error code is here when can't be started.)

0x09 →            'Gain of the camera'
0xa0 0x0f →       'Exposure time = 0x0fa0 (4000)'
```

```
    ← 0x02          Two stars has been found.
```

Now the user can get stars' data as follows:

```
0xca →          'AutoGuiding function'
    ← 0xca          Ok.
0x39 →          'Get star data'
    ← 0x00          Ok, allowed. (Error code is here when it's not possible.)

0x01 →              'Index of the star. We need the 2nd one.'
    ← 0x00          Ok, star data will be sent. (Error code is here if can't.)

    ← 0x78 0x01       X position is 0x0178 = 376
    ← 0x23 0x01       Y position is 0x0123 = 291
    ← 0xa5 0x02       brightness is 0x02a5 = 677
    ← 0x05            Star occupies 5 pixels.
    ← 0x9b            Peak pixel value is 155
```

If the star had saturated pixel, the data was like this:

```
    ← 0x78 0x01       X position is 0x0178 = 376
    ← 0x23 0x01       Y position is 0x0123 = 291
    ← 0xa5 0x82       brightness is 0x02a5 = 677 + bit 15 set
    ← 0x05            Star occupies 5 pixels.
    ← 0xf5            Peak (saturated) pixel value is 245.
```

After choosing one of the stars (say index 1 above) it can be set to be followed by the camera:

```
0xca →          'AutoGuiding function'
    ← 0xca          Ok.
0x3f →          'Get star data'
    ← 0x00          Ok, allowed. (Error code is here when it's not possible.)

0x78 0x01 →         The X position read previously.
0x23 0x01 →         The Y position.

    ← 0x00          Ok, guiding window has been set.
```

(In the case of the Camera is powered down (pulled out) meanwhile, a non-zero error code is returned as the last byte.)