LAMA Standard

Release 1.7.0

Visitech AS

Jun 12, 2025

CONTENTS:

LAMA Standard (LUXBEAM Additive Manufacturing App), hereafter LAMA, is used to easily control multiple photoheads produced by Visitech. It offers high-level functionality for imaging (static and scrolling), light source control, error checking, maintenance and much more. The API uses a TCP text protocol with JSON formatting and a binary protocol where binary data is required.

LAMA is supported on Windows, Linux, and MacOS operating systems. Platform support is AMD64 and ARM.

Currently supported Visitech products are:

- LUXBEAM 9500 based
- LUXBEAM 4800 based
- LUXBEAM 4600 based
- LRS MCx HD V1 (4600)
- LRS MCx HD v2 (4800)
- LRS MCx WQ (9500)
- LRS MCx 4K (9800)
- LRS Compact
- Some emulated devices for development and testing purposes

Please be sure to read this whole documentation from top to bottom before trying to operate LAMA. If you do not do that then remember these very important points:

- **Before starting to use LAMA Standard it is REQUIRED to adjust the MTU of the network interface card to 9000 bytes (jumbo frame). Failure to do this will lead to an inoperable projector. This does not apply to LRS-Compact.**
- **For Luxbeam 4800 and Luxbeam 9500 based products it is required to open firewalls for incoming UDP requests on port 52987. For consumer Windows OS this means setting the network to being "private" instead of "public".**

Start with this section: *Network Configuration*

CHAPTER

ONE

NETWORK CONFIGURATION

LAMA uses Ethernet and UDP/IP to communicate with the photoheads so before starting it is important to understand a few things that are required by the network between the client PC and the photohead.

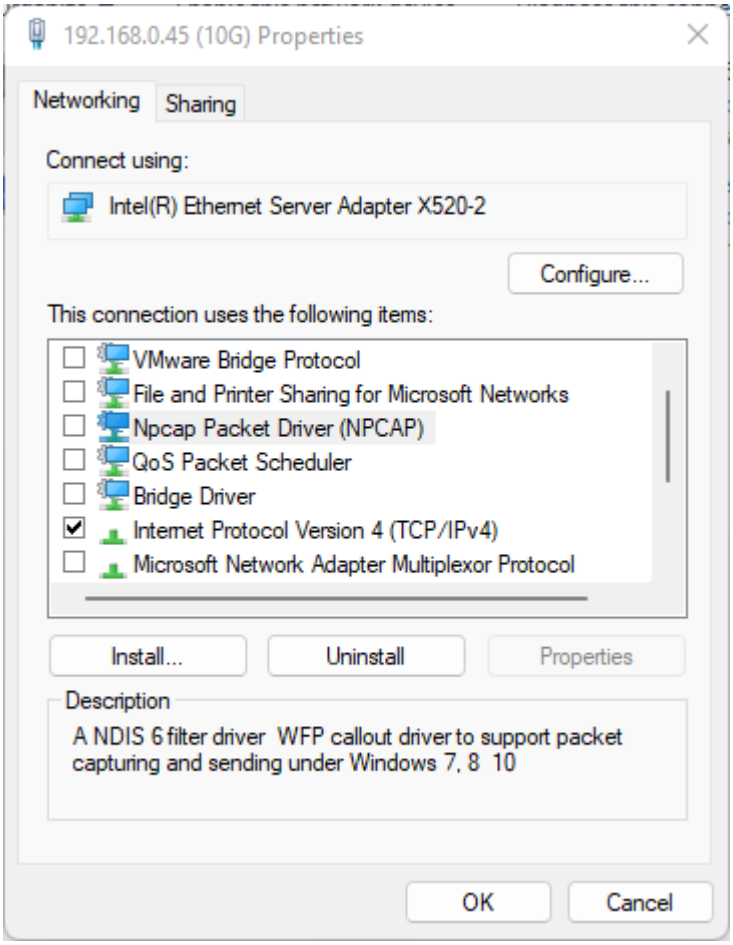
1.1 Hardware Requirements

- One network interface card per photohead
- Network interface card must support a maximum MTU of 9000 bytes
- No switch or router is allowed between client PC and photohead
- Cat.5e grade cable for 1 Gigabit RJ45
- Mellanox (NVIDIA) or Intel card with compatible transceivers is recommended for 10 Gigabit SFP+
- For 10 Gigabit we recommend Twinax passive cable if length is shorter than 7.5M, and active fiber cable above that

1.2 Operating System Requirements

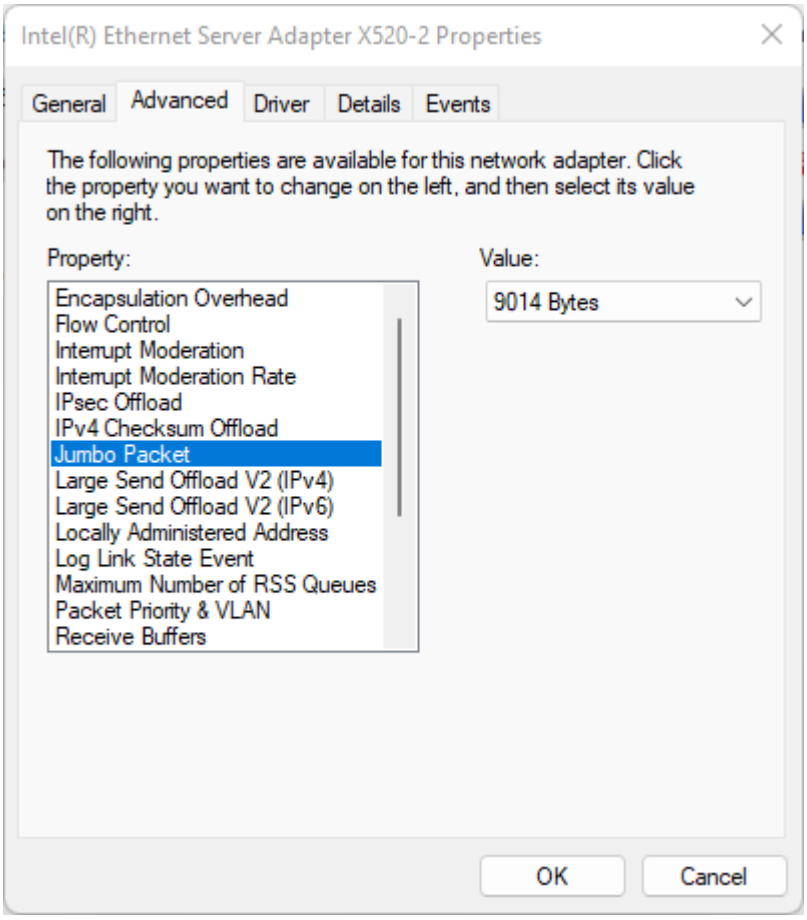
- No packet sniffer should be used, e.g Wireshark/pcap on the network interface card connecting the photohead to the client
- When using a VM to connect to the photohead, e.g running LAMA on a Ubuntu VM on a Windows host, we can not guarantee connection
- MTU must be set to 9000 (DOES NOT APPLY TO LRS-COMPACT)
- Any interrupt moderation must be disabled to get maximum performance for image data transfer
- On Windows all services other than IPv4 should be disabled on the network interface
- Port 52987 must be opened for incoming UDP requests for products using Luxapp.
- For Windows the photohead network interface setting must be a Private network and not a Public network.

1.2.1 Only IPv4



Make sure that all options are unchecked except IPv4 in network interface properties for Windows.

1.2.2 MTU (Jumbo)



Make sure that MTU is at least 9000B (jumbo packet). If you are connecting to a LRS-COMPACT, the MTU should be set to 1500B.

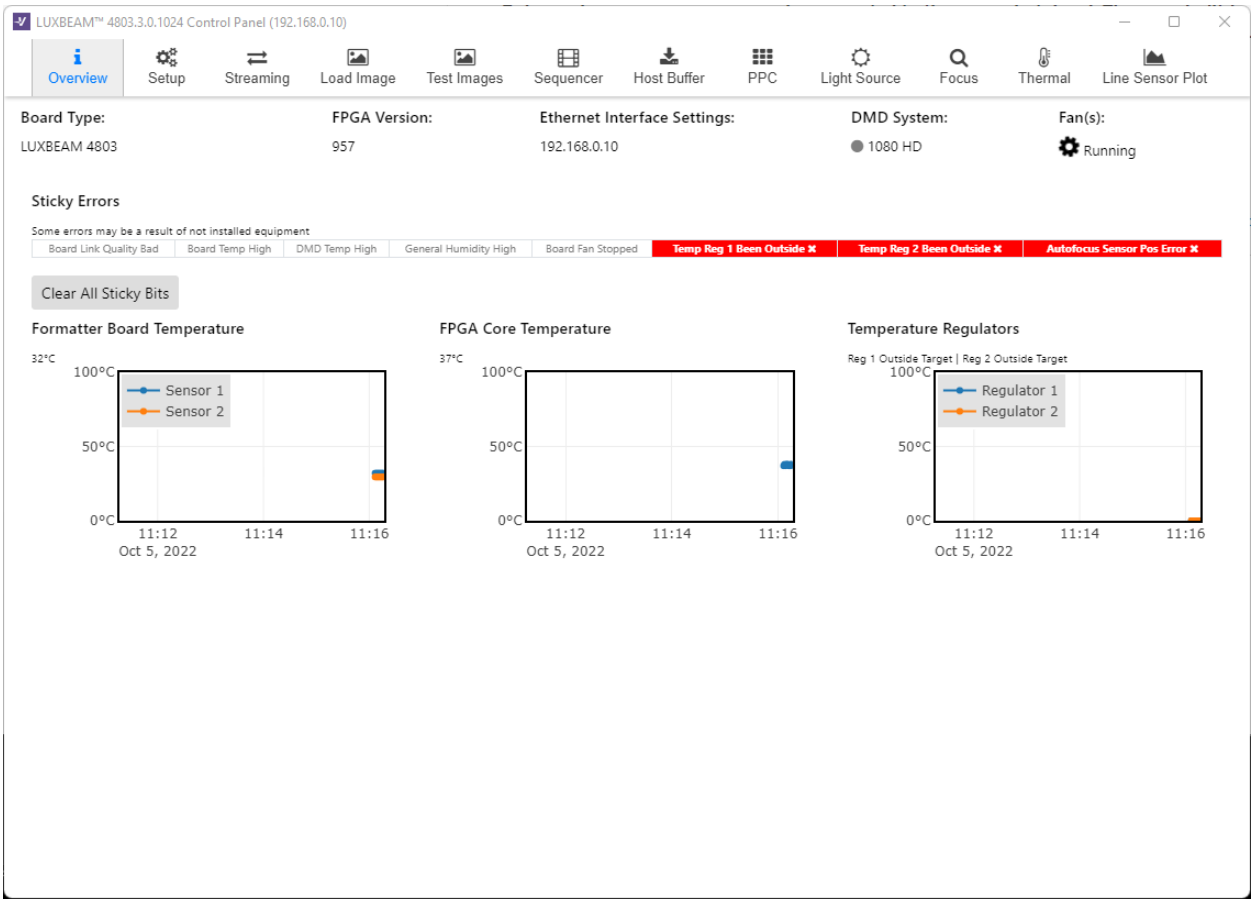
CHAPTER

TWO

LUXAPP

**Only applies to LRS-MCX HD V2, LRS-MCX-WQ and LLS products. If you are using another product you should skip this chapter. **

Luxapp is a service and a GUI that runs on the client PC controlling the FPGA in the photohead which handles all of the low-level register reads and writes. It also controls the flow of the image data being streamed to the photohead in realtime. Luxapp is used to control/configure one photohead outside of LAMA. One Luxapp thread/process will be started for each photohead by LAMA Standard automatically.



Luxapp's Control Panel overview GUI

The hierarchy is simple: LAMA speaks to Luxapp and then Luxapp speaks with the photohead.

Only one Luxapp process can ever be connected to the same photohead. There are built-in locks that should prevent this from happening, but in some cases such as running one in a VM and one on the VM host it is not

possible to detect. This state will cause undefined behaviour.

Some operations can not be done from LAMA and using the Luxapp GUI is required (described in the "Get Started" section):

- Changing IP of the photohead
- Changing MAC of the photohead
- Setting Formatter Operation mode (e.g static/direct mode required by LAMA Standard, SPX required by LAMA Pro)
- Upgrading firmware on the photohead

2.1 Usage

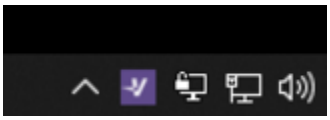
Luxapp is available in all LAMA packages, and is required for running LAMA. It can be found in the subdirectory "resources/luxapp". Starting Luxapp with the GUI can be done by running luxapp_tray.exe.

Luxapp takes all its arguments for startup as command line arguments.

The following arguments are available for "luxapp_tray.exe":

```
--ip <192.168.0.10> IP address(es) for photohead(s). Comma seperated.
--port <52985> Port(s) to bind the Luxapp server application(s) to. Comma seperated.
```

The default IP is 192.168.0.10 if no arguments are provided when launched. Once started an icon for the service will show:



Luxapp Service Tray icon

Right click this icon to open a menu and the option of control panels for any of the photoheads connected.

2.2 Command-line Arguments

For Luxapp we offer some command-line operations for things such as updating firmware, to facilitate quicker upgrade of machines with a large amount of photoheads, or during the initial setup.

To run these special commands we use only the Luxapp server executable "luxapp.exe", not the Tray with the graphical user interface.

e.g

```
.\luxapp.exe 192.168.0.10 --flash-on-mismatch
```

Available arguments:

```
--flash-bitstream <path to bitstream> (force a specific firmware to be flashed␣
↪immediately)
--flash-on-mismatch (flash supported firmware and then exit, exit code 0 on success)
--check-fw (exit code 1 if needs upgrade, otherwise 0)
```

For –flash-bitstream compatible firmwares can be found under sub-directory **resources/bitstream/<product>/**

- **s0029_lb4k8_grot.bit** - Luxbeam 4800 SPX Mode (LAMA Pro)
- **s0033_lb4k8_spx1_no_morph.bit** - Luxbeam 4800 Static/Direct Mode (LAMA Standard)

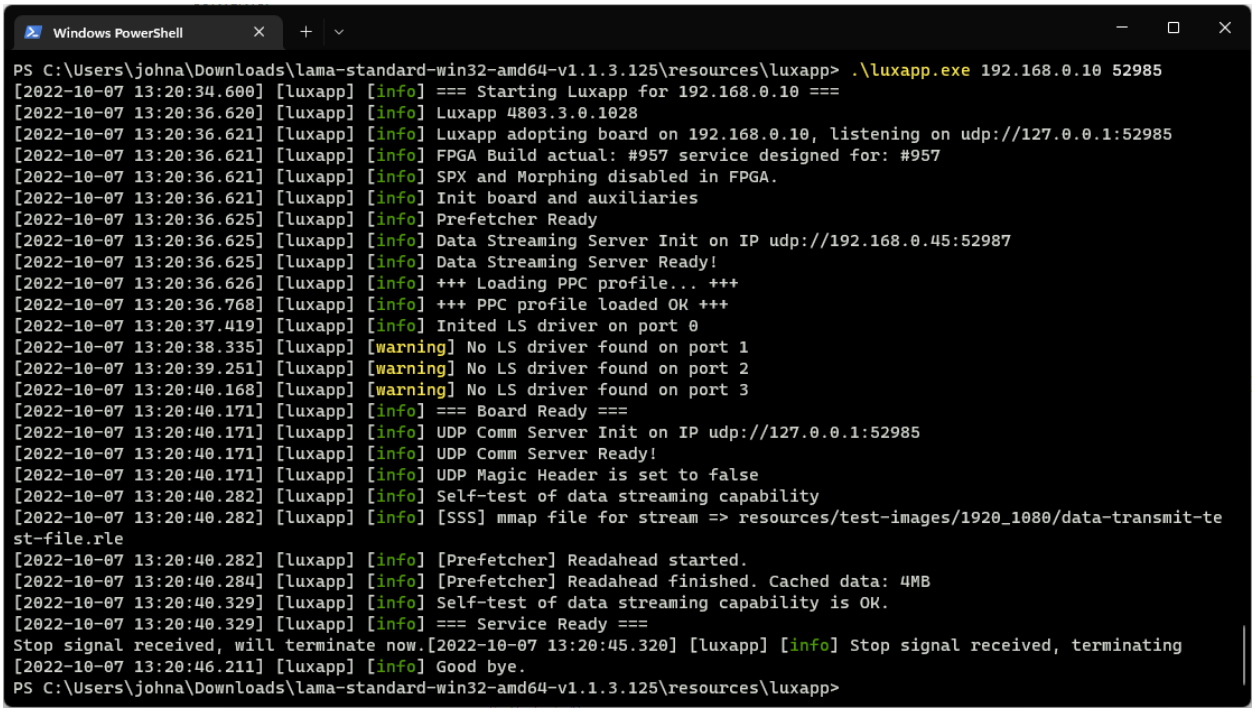
Do not power off or disconnect the photohead while it is being reflashed.

2.3 Troubleshooting

Sometime it can be useful to read the terminal output of "luxapp.exe" (not "luxapp_tray.exe") when trying to troubleshoot different issues.

Run:

```
.\luxapp.exe <ip> 52985
```



CHAPTER

THREE

GET STARTED

Extract the contents of the LAMA package into a directory that you have access rights for.

3.1 Is the Photohead Ready?

The photohead should be powered up.

For MCx:

"Initialise" LED should be solid yellow. "Ready" LED should blink green. If the indicator LEDs show a different pattern please contact Visitech for support.

3.2 Physical Connection

Connect the photohead to the client PC using a RJ45 CAT5e cable or SFP+ compatible cable.

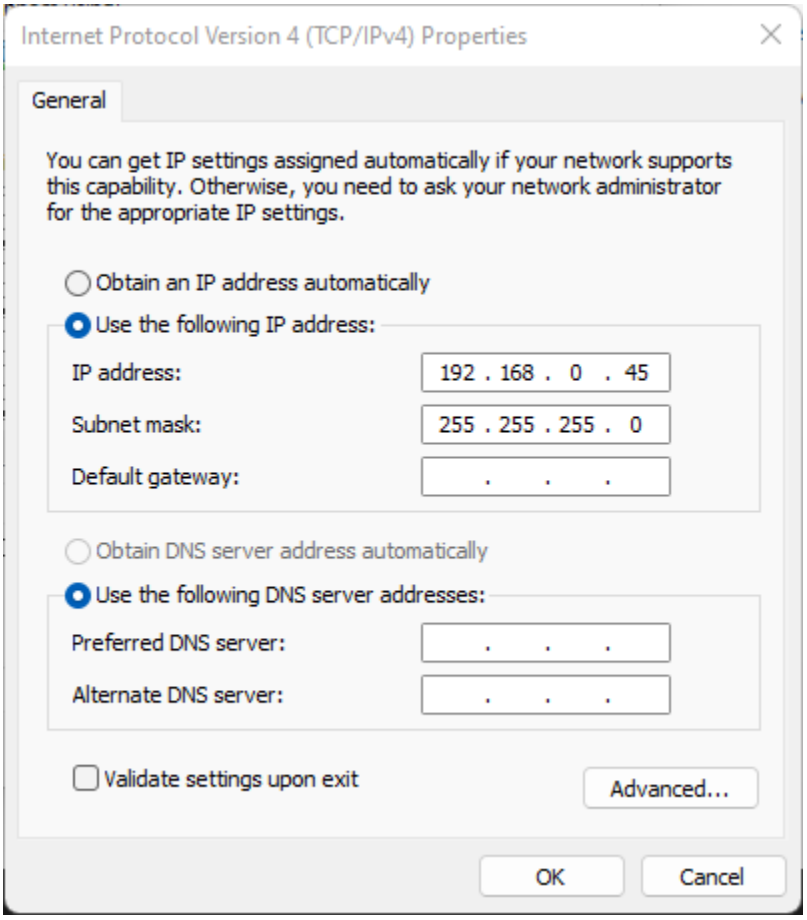
NEVER CONNECT BOTH, IF YOU DO CONNECT BOTH, REMOVE ONE AND RE-POWER PHOTOHEAD.

NOTE: When switching between interface card(s) or port(s) it may be required to re-power the photohead because the FPGA will bind to its target IP after the first packet is received from the client.

After connecting the cable, make sure that the interface detects a link to the photohead as indicated by LEDs on the network interface card.

3.3 Setup of Network

Make sure that you have read the section about network configuration and requirements before starting. Setting the MTU size => 9000 (does not apply to LRS-COMPACT), and disabling of other services for the interface are very important.



Setting the IP of your network interface

We need to setup the interface for the photohead. When a new photohead is received it will be bound to the default IP: 192.168.0.10. You will need to set a manual IP address. There is no DHCP service for the photohead, so it is required to define the IP as shown in the screenshot. The client PC can use any available IP in the network subnet.

Make sure not to set the client PC IP to be the same as the photohead (not 192.168.0.10 as shown in the case above)

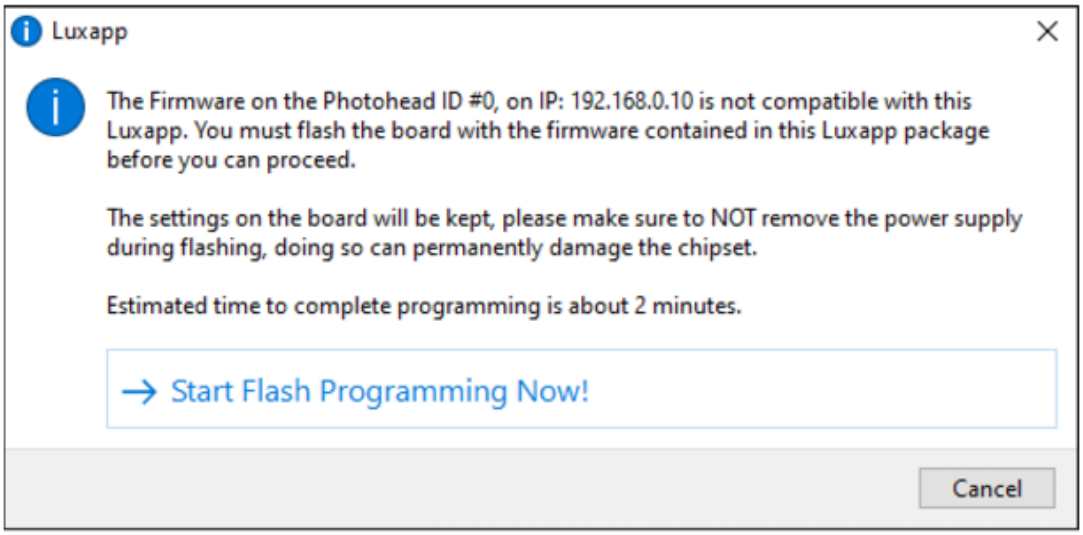
Check that the interface card is shown as connected in the OS and check that the indicator lights on the photohead are lit.

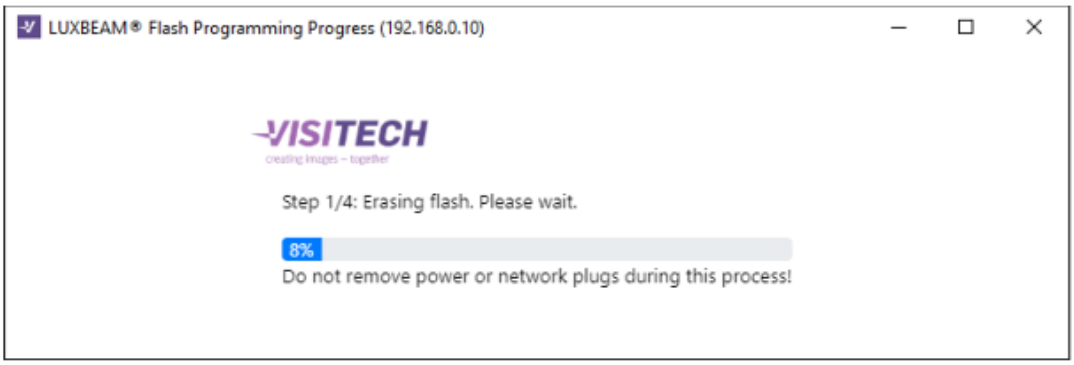
3.4 Start Luxapp

Only applies for 4600, 4800, 9500 based products: As described in the section about Luxapp, please start Luxapp (luxapp_tray.exe).

3.5 Firmware Update

Depending on the age of the firmware loaded on the photohead, it may be required to do a firmware upgrade. If so, you will be prompted automatically when you try to start luxapp_tray.exe.



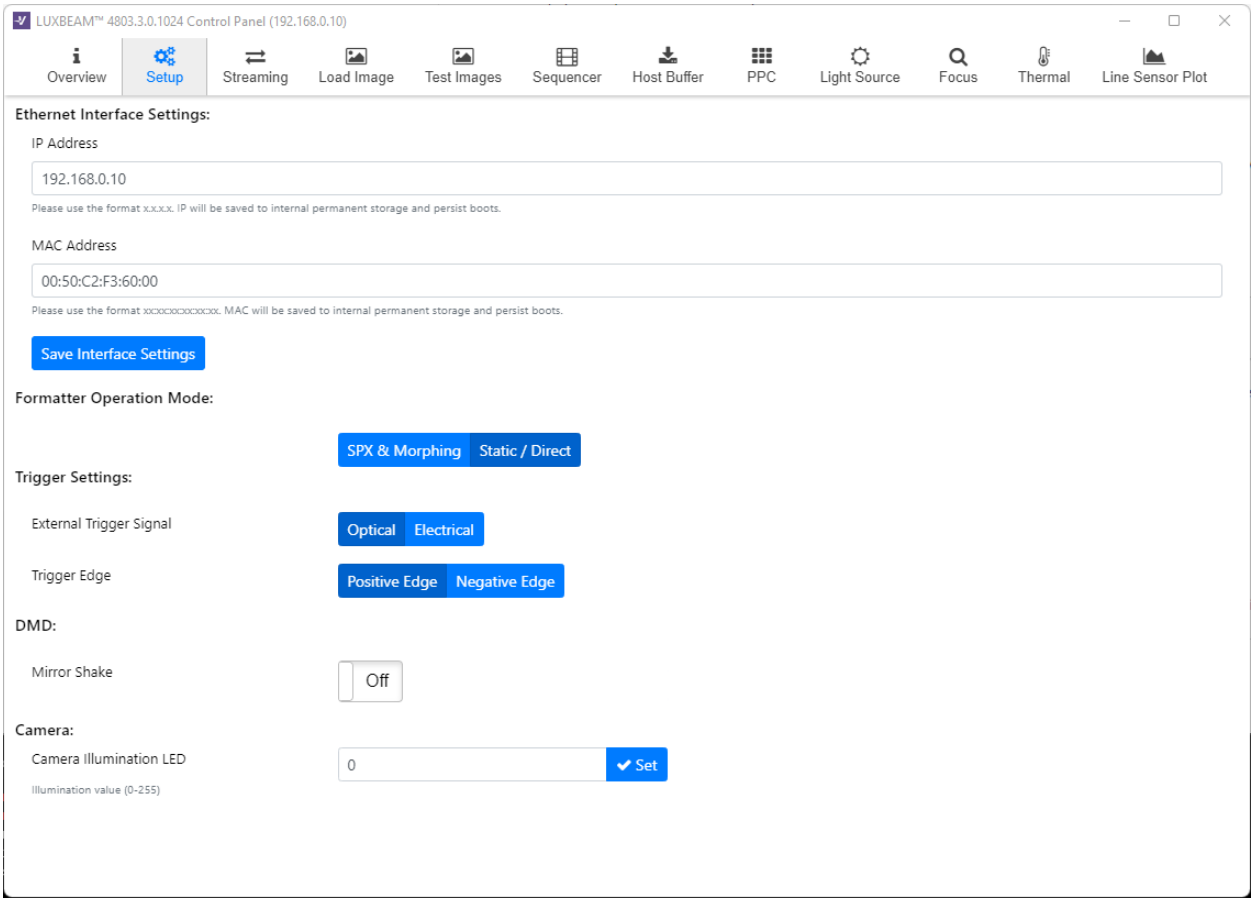


The dialog prompting to upgrade the photohead.

Please follow all instructions given by the dialogs carefully. It is important to not remove the power or network connection while the process is ongoing.

The firmware on the photohead is bound to the LAMA version used so switching between different LAMA versions might cause either a downgrade or an upgrade of the firmware. Both scenarios are supported.

3.6 Initial Setup of Photohead



Settings for photohead in Luxapp Control Panel

When a new photohead is received, it will be bound to the default IP: 192.168.0.10. If you have multiple photoheads this will need to be changed to support addressing of individual photoheads. This is done in the "Setup" tab in the control panel of Luxapp. This initial setup must be done before starting LAMA Standard.

Also make sure to check that the MAC address on the photoheads are unique. If not, you will need to change them as well. After changing MAC it might be a good idea to reboot your operating system. It might confuse the operating system when devices swap MAC in realtime.

It is recommended to physically label the IP address on the phothead to avoid forgetting the IP that was set for it. After setting with the "Save Interface Settings" button the photohead will no longer be reachable on that IP. After setting the IP, the connection may be lost to the photohead. You will need to update the interface settings in Windows before it can reconnect.

For LAMA Standard it is required to select "Static / Direct" formatter operation mode. Changing this mode will often require a firmware upgrade again. This is normal. Follow the instructions given by the dialogs.

3.7 Test Streaming

- In the control panel click the "Load Image" tab and select 1-bit .BMP encoded.
- Click "Streaming" tab.
- Click "Browse" button.
- Navigate to the "example" subdirectory in the LAMA package. Open "example/stripe-files/1920x20000-1bit-inverted.bmp".
- Click "Set Offered Stream"
- In the field called "Rows to Load" fill in 19000.
- Click "Set Morph Request"

•In the statistics below check the following values:

  - Lines Ready To Print: 19000
  - Bytes Transferred: 9600000B
  - RX CRC: 0x31261b
  - TX CRC: 0x31261b

If everything is correctly setup, these values should match yours.

3.8 Factory Reset Button

On the back of all photoheads there are small holes with reset buttons.

One to soft reset the photohead. One to restore factory settings, including setting IP to 192.168.0.10. This way, if you ever forget what IP has been programmed into the photohead, this button can be clicked (do not keep it pressed). Do not use much force , the button is a small.

3.9 Troubleshooting

If Luxapp reports any error when being launched please open the luxapp.log file and read it and look for hints.

- Failing self-test: This happens when the connection between the client PC and the photohead are not setup correctly and/or the physical link quality is poor. Check the section about Network Configuration and requirements.

CHAPTER

FOUR

LAMA SETUP

Start by extracting the contents of the LAMA package into a directory that you have access rights for if not already done. Both LAMA and Luxapp will create log files in their working directories so make sure you have write priveleges as well. The log files are cyclic. Once the file size reaches 10MB, they are cycled so that the oldest log file is deleted. This is to avoid large amounts of log data persisting on the system.

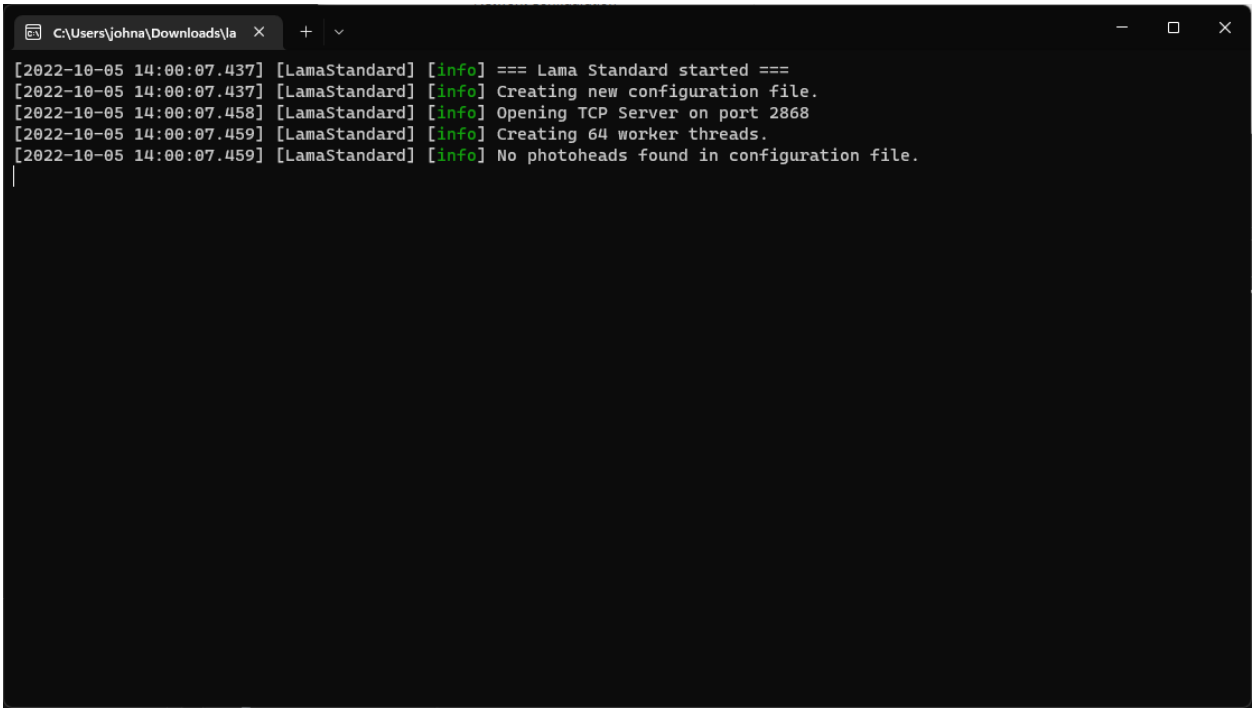
Make sure that all the photoheads for the system have been setup, upgraded and verified as described in the "Get Started" section.

4.1 Start LAMA Server

The first step is to start LamaStandard.exe

Available arguments for LamaStandard.exe are:

```
--listen-port <default 2868> Port to bind the LamaStandard server application(s) to.
--log-path <default current working directory> Path to use as log directory.
--log-level <default level 2 (info)> 0-4: 0 - TRACE, 1 - DEBUG, 2 - INFO, 3 - WARNINGS,␣
↪4 - ERROR
```

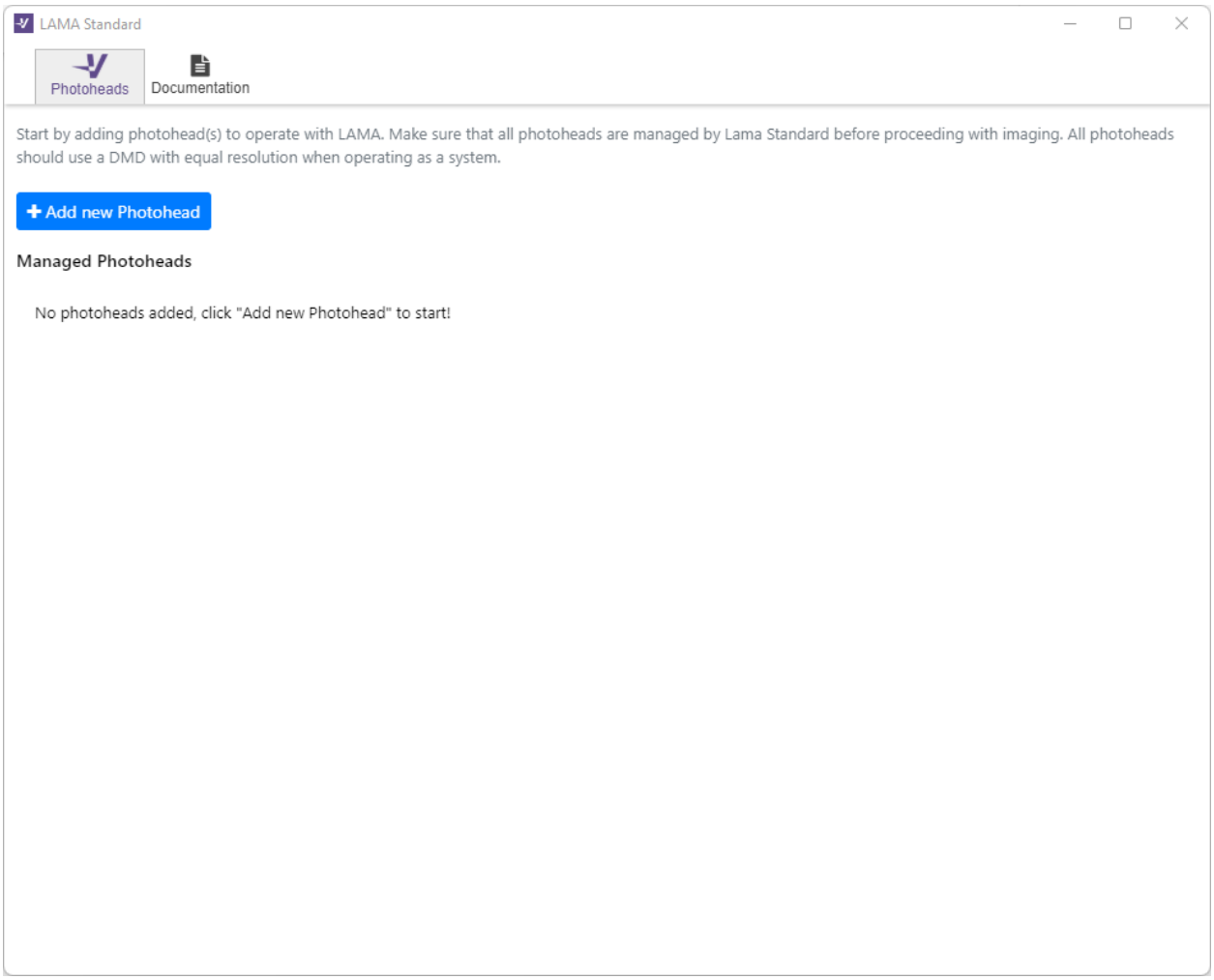


Initial startup of LAMA

Note the log line indicating that no photoheads are found in the configuration file.

4.2 Start Test Client

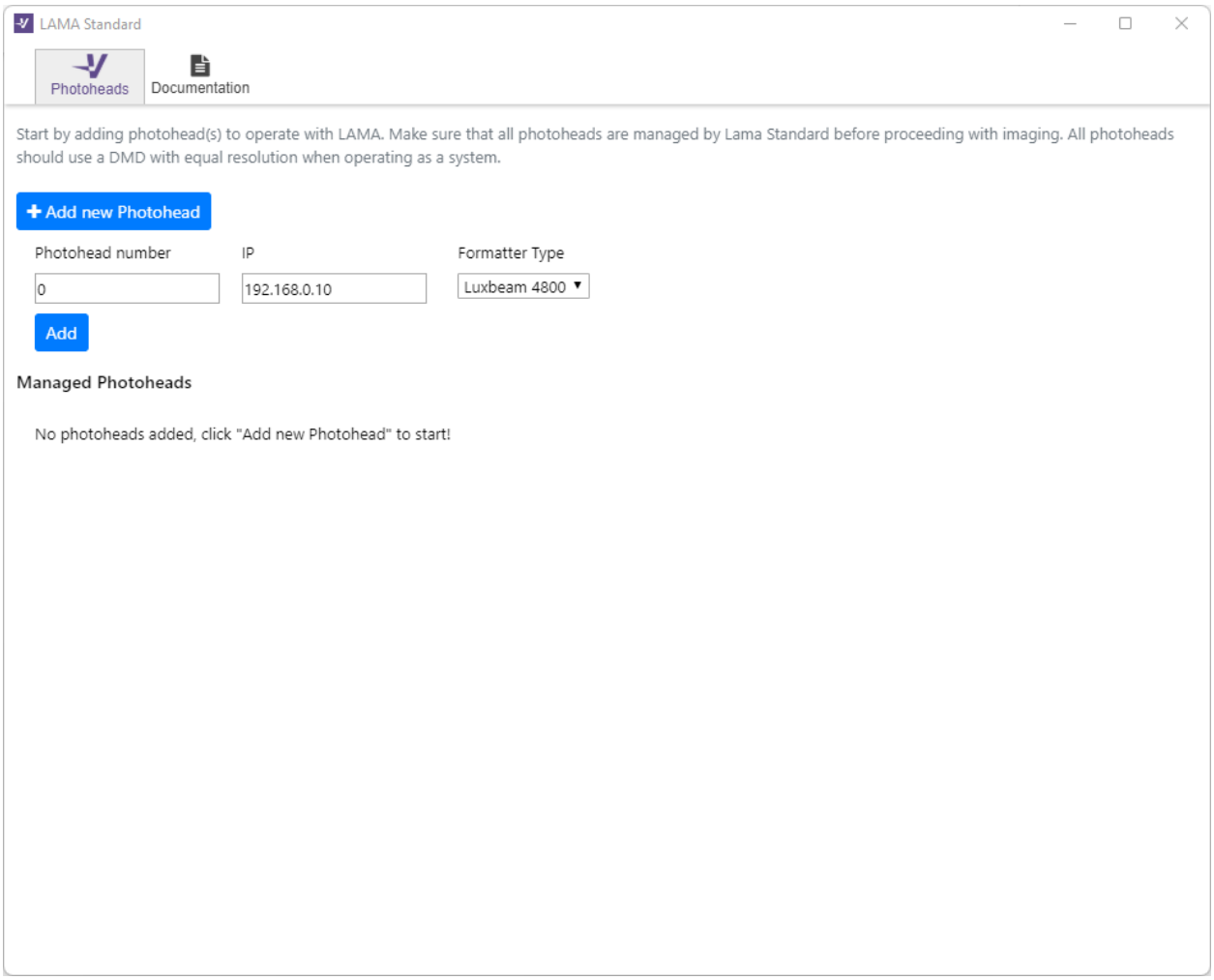
LAMA includes an app with a graphical interface to test basic operation of the LAMA Server. This is not meant to do advanced operations since typical use of the system varies wildly. The test app/client is located in the LAMA package under subdirectory “client”. Open LamaStandardClient.exe. **Note: The server must be running at this point.*



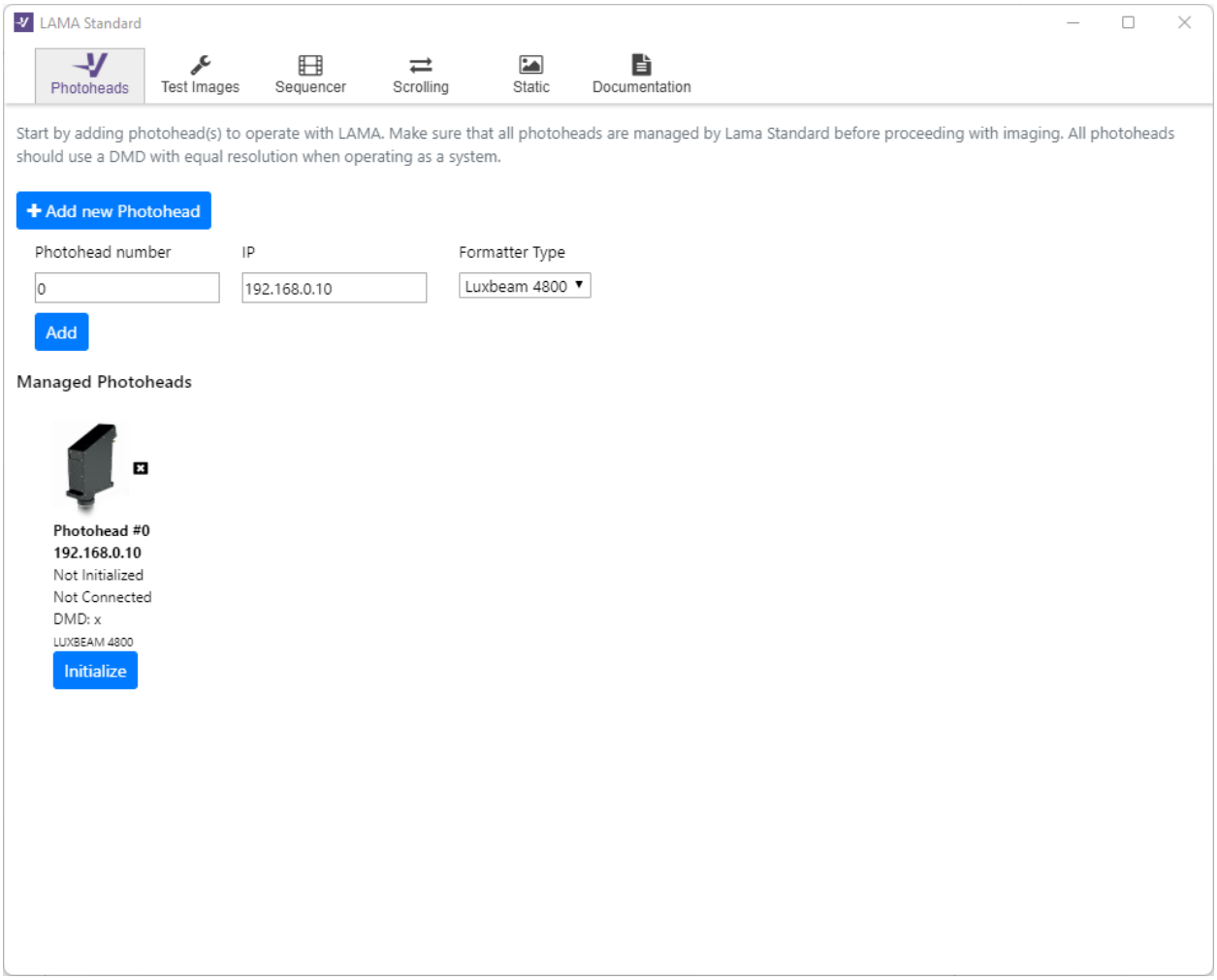
LAMA Standard Client initial start-up

Notice no photoheads have been added to the pool of managed photoheads. Click “Add new Photohead”. Input boxes for “Photohead number”, “IP” and “Formatter Type” will appear. Fill in these before clicking “Add”.

Note: Photohead number can be any number and is used to later address that specific head. In order to keep this simple we recommend starting at 0 and increase the number by one for each new photohead.

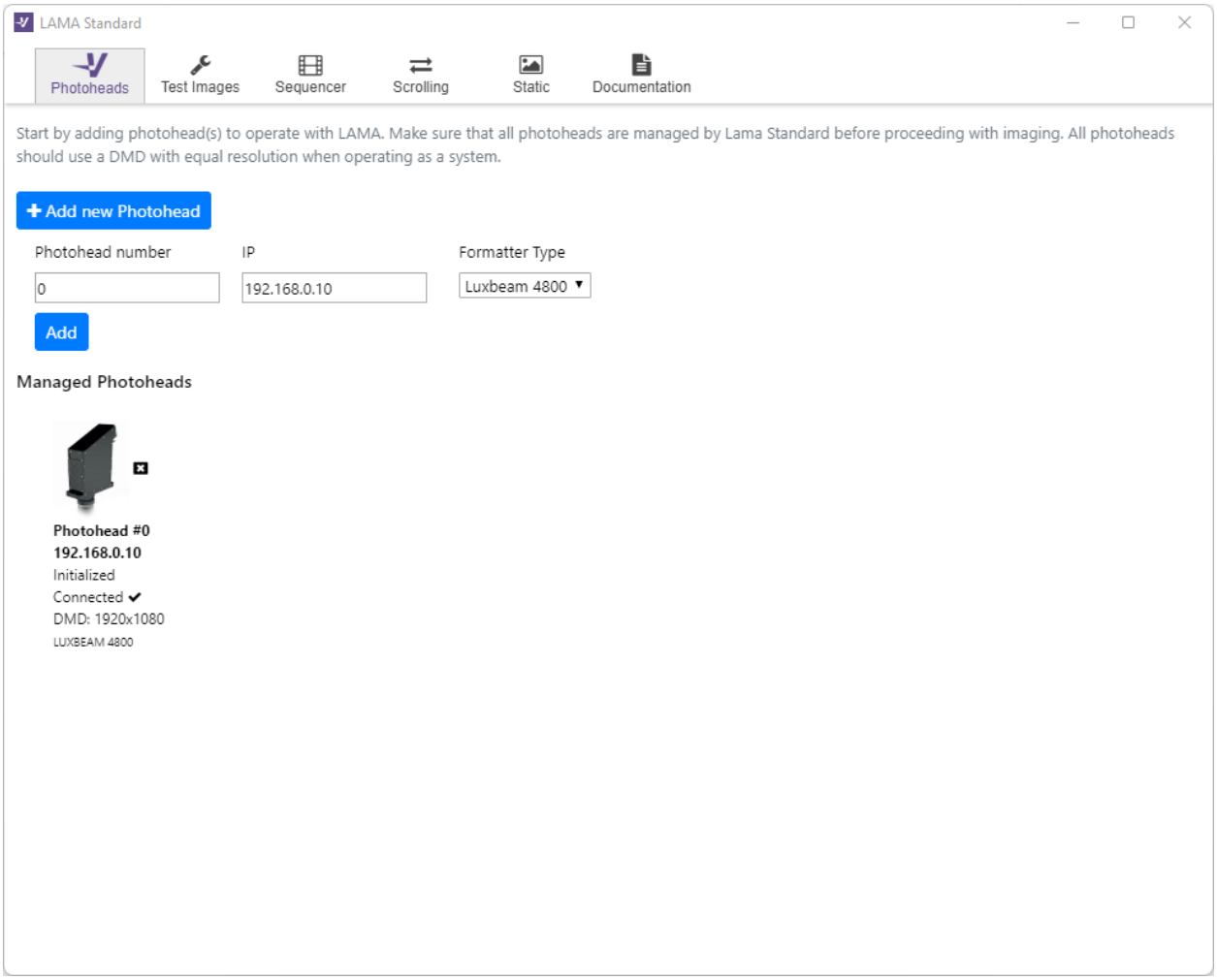


Add photohead details



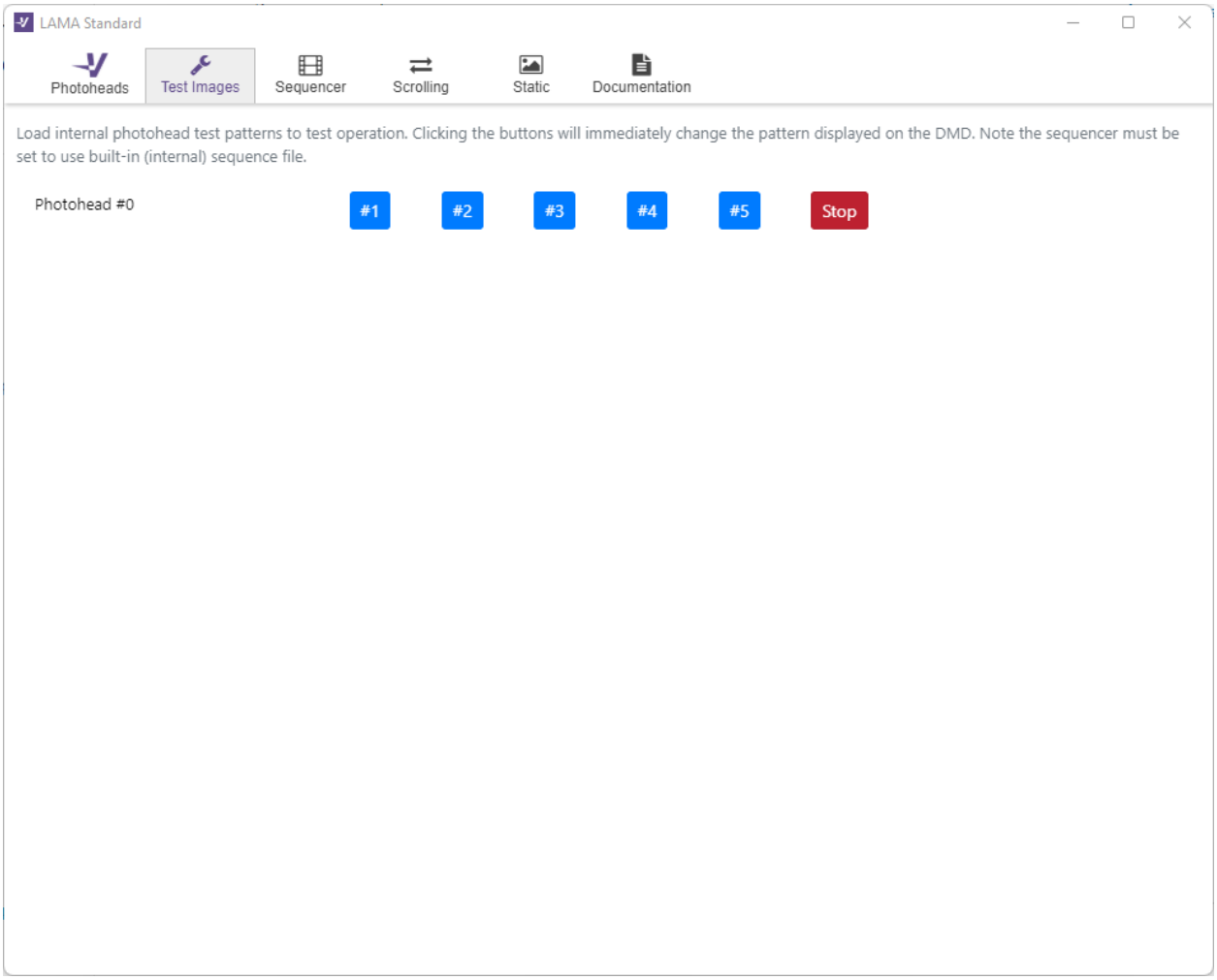
Photohead added

Then we must "Initialize" the photohead. If this fails please check the log file or the server terminal for any error messages which will point out the cause for the fail.



Photohead successfully initialized

We can load and project a built-in test image to test basic funtionality. **Note: Wear appropriate eye protection when projecting images.** Click the "#1". The selected photohead will immediately project a test image.



Load test image

If not light is emitted from the photohead, you have to check that the “door open” safety circuit is connected to the photohead.

4.3 Persistent Settings

Once a photohead has been added to the LAMA Server, its details will be saved in the settings.json file in the working directory. The next time the LAMA Server is started it will automatically start with the photoheads added previously. They will also be initialized automatically so it is not neccesary to add photoheads that are already in the “pool”.

4.4 Initialization

Initialization of photoheads is a required part of LAMA. It involves checking and modifiying settings on the individual photoheads to be appropriate for LAMA. There are also some self-tests being run to check if everything is working as expected. Photoheads need to be initialized to do other operations. Only the photoheads that are involved in an operation will have to be initialized.

During initialization, one Luxapp process is started per photohead. To make sure that there are no duplicate Luxapp processes running, any existing processes will be killed when LAMA is started. This is because there can only be one Luxapp running per photohead at any time. Make sure that any unsaved changes in Luxapp GUI have been saved before starting LAMA.

CHAPTER

FIVE

OVERALL PROTOCOL DESCRIPTION

The API protocol **called Lucom, "Luxbeam Common API"** is based on JSON text and/or BSON over TCP.

JSON is described here: https://en.wikipedia.org/wiki/JSON.

BSON is described here: https://en.wikipedia.org/wiki/BSON.

5.1 TCP Server

The application exposes its API through TCP port 2868. You may change the port by starting the application with the option –listen-port <port>.

5.2 JSON API

All requests must conform to the UTF-8 text encoding.

All JSON API requests start with the text sequence "VT-JSON\r\n" then the actual JSON encoded request object must be written, followed by an ending of "\r\n\r\n" (CRLFCRLF), two line shifts.

The response once the request has been completed is returned in the same format with the VT-JSON header, CRLF, the JSON response and delimiter (of CRLFCRLF).

Example of a simple request:

Request:

```
{
  "_id": 1,
  "version": 1,
  "module": "LightSource",
  "cmd": {
    "func": "SetAmplitude",
    "args": {
      "ph_no": 0,
      "led_no": 0,
      "amplitude": 1000
    }
  }
}
```

Successfull response:

```
{
  "_id": 1,
  "status":"ok",
}
```

5.2.1 Structure

Some keys are always required in the request object, and some keys will always be present in the respose coming back from the server.

For requests these are:

- module - Describing the module to access.
- cmd - A dictionary describing the command to start and its arguments.
  - func - The function name to execute.
  - args - Any arguments the function requires.

Optional keys for requests are:

- _id - A userdata number to match the response object to the request sent.
- version - If a specific version of a request is required (where applicable).

For responses the keys are:

- status - This indicates the status of the command.
- ret - *OPTIONAL.* Functions that return data back to the user will always be encapsulated in a "ret" object.

If the command has failed with the "status" key being "fail" there are additional keys that will always be present:

- ret - A dictionary consisting of
  - exception_message - A textual explanation of why command failed.
  - exception_code - A number indicating what roughly failed. Possible codes described in *Exception Description*.

Some commands that take a long time can return the keyword "working" together with a progress indication as described in the API description. You can still issue commands and get other responses while it is working to finish.

Furthermore commands that take a long time can be canceled by issuing a stop request object with the same _id as the initial command started:

Request:

```
{
  "_id": 1,
  "action":"stop",
}
```

Response:

```
{
  "_id": 1,
  "status":"stopped",
}
```

5.2.2 Examples

The application is bundled with some Python test programs to familiarize the user with the interface. It is recommended to review these examples to understand how to make a TCP client read and write request over the TCP IPC channel (Inter Process Communication).

All examples can be found in the package under the directory "example".

5.2.3 Parallelism

The application operates asynchronously and multi-threaded. That means that you can send multiple request objects and they will be executed at the same time. E.g starting commands on different photoheads at the same time. The user must then use the _id field to tie request to responses which may arrive in a different order than executed, due to different execution time. There are locks in place to avoid allowing the user to do things that will collide, so some resources can not be modified at the same time. It is encouraged to send commands in bulk to increase the throughput. In cases where it is required that things happen in a serialized fashion, an array of request objects can be sent in. They will then be executed one by one as described in the list.

Example of this operation:

Request:

```
[{
  "_id": 1,
  "module": "LightSource",
  "cmd": {
    "func": "SetAmplitude",
    "args": {
      "photoheads": [0,1],
      "led_no": [0,1],
      "amplitude": 1000
    }
  }
},
  "_id": 2,
  "module": "LightSource",
  "cmd": {
    "func": "StartStop",
    "args": {
      "photoheads": [0,1],
      "no": [0,1],
      "toggle": true
    }
  }
]
```

Response (one response object will be delivered per command in the array above):

```
{
  "_id": 1,
  "status":"ok",
}

{
```

(continues on next page)

(continued from previous page)

```
  "_id": 2,
  "status":"ok",
}
```

If you do not send the request objects as an array, they will be executed arbitrarily in time. First in does not mean first out. This depends on the time it takes to execute the command. Often it does not matter in what particular order the commands arrive or are executed.

5.3 BSON API

All the commands that support JSON can also take BSON request. **BSON is needed whenever we want to transmit binary data, as JSON only supports text. Failure to respect this will result in a error. All functions that require BSON is marked with a warning.**

The formatting of a BSON request is the same as for a JSON request, except for the header. All BSON API requests start with the text sequence "VT-BSON\r\n" then the BSON binary blob object must be written, followed by an ending of "\r\n\r\n" (CRLFCRLF), two line shifts.

The return will be in the same format, and follow the same scheme as for JSON.

CHAPTER

SIX

SEQUENCER

The sequencer is a small 16-bit CPU that is responsible for selecting data to load on the DMD. The goal of the sequencer is to provide a flexible platform that can be used for many different applications. The sequencer language is translated to machine code automatically when loaded from the application.

Not all products have the sequencer available. Product that do support it are:

- LRS MCX V1
- LRS MCX V2
- All LLS products

Products that does NOT support it are e.g:

- LRS MCX 4k
- LRS MCX 8k
- LRS MCX Compact

Trying to use the sequencer API commands on a product that does not support it will result in an exception returned.

6.1 Programming Language

The sequencer language is a proprietary language designed by Visitech to best fit the requirements. It is case sensitive. The sequence contents should be saved to a text-file. Each command must be written on a single line in the text-file. Commands and arguments on a single line must be separated by one or more spaces or tabs.

Empty lines are ignored.

Comments can be put on separate lines or after a command with arguments. A comment starts with the character '#' and is ignored by the parser. Arguments that are numbers may be written in decimal or hex. Hex number must be prefixed with «0x», e.g. the decimal number 100 is written as «0x64». Negative numbers (valid for the «Add»-command only) is written as decimal, e.g. «-10».

The sequence file is uploaded to the board using the API command *Sequencer/Load.*

Most sequencer commands have a last argument, WAITFOR. This is used to indicate how many ticks should elapse before the next command is executed. All commands take a minimum of 1 tick (ClearAll must have minimum 16 tics), hence 1 is the smallest allowable value for WAITFOR. One tick = 250 ns.

Note that the last line in a sequence file must be a «Jump» or «JumpIf» command.

6.2 Example

An example where we load 1080 rows (a full DMD height for a HD DMD) from the first frame in the host-buffer. This will result in a static picture on the DMD that will not change.

```
#-------------------------------------------------------------
# Command         Options                              Waitfor
#-------------------------------------------------------------

AssignVar         MemStartRow 0                          1
AssignVar         DmdStartRow 0                          1
AssignVar         Inum        0                          1
AssignVar         NumRows     1080                       1

# Loop
Label             StartHere                              1
LoadRow           DmdStartRow NumRows Inum MemStartRow  173  # (1080*40ns)*(4tics/us) =␣
↪172.8
ResetGlobal                                             20
LightPulseWord    15                                    27
Jump              StartHere                              1
```

6.3 Command Description

Note that parameters that are expected to have a number (N) as input, may be replaced by an alias name. This requires that the alias has been defined using the command «Alias» before referring to it. Parameters that can have a variable (V) as input must have the variable assigned using «AssignVar» or «AssignVarReg» before using the variable.

6.3.1 Load Commands

Command group that load contents to the DMD memory (not effectuated to the DMD-mirrors)

LoadGlobal

Load the complete image from inum INUM to the DMD.

Parameters:

- INUM <V> - Image number offset in host-buffer.
- WAITFOR <N>

LoadSingle

Load a single group, starting at GRP from inum INUM. The start location in the inum where the data is taken from is: inum_start + GRP*no_of_lines_per_group.

Parameters:

- GRP <N> - DMD
- INUM <V> - Image number offset in host-buffer.
- WAITFOR <N>

LoadDual

Load two groups, starting at GRP from inum INUM. The start location in the inum where the data is taken from is: inum_start + GRP*2*no_of_lines_per_group.

Paramters:

- GRP <N> - Can have values 0, 2, 4, 6, 8, 10, 12, 14
- INUM <V> - Image number offset in host-buffer.
- WAITFOR <N>

LoadRow

Load the given number of rows, NO_OF_ROWS from the row MEM_START_ROW of inum INUM to the DMDs row DMD_ROW_OFFSET.

All parameters, except for WAITFOR, are variables that must be predefined.

Parameters:

- DMD_START_ROW <V> - Row to start loading into.
- NO_OF_ROWS <V> - Number of rows to load.
- INUM <V> - Image number offset in host-buffer.
- MEM_START_ROW <V> - Row offset in host-buffer.
- WAITFOR <N>

ClearSingle

Clear the DMD memory (fill with black) corresponding to the group GRP.

Parameters:

- GRP <N>
- WAITFOR <N>

ClearGlobal

Clear the whole DMD memory (fill with black).

Parameters:

- WAITFOR <N> - Note: minimum 6 ticks.

6.3.2 Reset Commands

(Not to be confused with traditional meaning of «reset») Put the DMD mirrors into position given by contents of DMD memory.

ResetGlobal

Reset all groups, i.e. display the loaded contents on the DMD.

Parameters:

- WAITFOR <N>

ResetSingle

Reset the group GRP, i.e. display the loaded contents of group GRP on the DMD.

GRP must be a numerical value.

Parameters:

- GRP <N> - DMD group
- WAITFOR <N>

ResetDual

Reset the dual group with start address GRP, i.e. display the loaded contents of the dual group starting at GRP on the DMD.

GRP must be a numerical value.

Parameters:

- GRP <N> - DMD group start, can have values 0, 2, 4, 6, 8, 10, 12, 14
- WAITFOR <N>

6.3.3 Assignment and Flow control

DeclareLabel

This command is used to forward declare a label that is to be used later. This is necessary when using a «JumpIf» command that will jump forward to a label. This label needs to be declared using «DeclareLabel» before the «JumpIf» command.

Max length of LABEL: 20 characters.

Max number of labels: 32

Parameters:

- LABEL - name of the label

Label

Define a label which is used as a target for the Jump command. LABEL must contain alphanumeric characters (a-z, A-Z) and may contain numbers (0-9). LABEL cannot start with a number.

Max length of LABEL: 20 characters.

Max number of labels: 32

Parameters:

- LABEL - name of the label
- WAITFOR <N>

AssignVar

Define a variable VAR and assign a value VALUE or var VAR_B to it. If the 2nd parameter is a variable, VAR_B must be defined. VAR must contain alphanumeric characters (a-z, A-Z) and may contain numbers (0-9). VAR cannot start with a number. Max length of VAR: 20 characters.

Max number of variables (including those assigned using AssignVarReg): 32

Parameters:

- VAR - name of the variable
- VALUE/VAR_B <N/V> - value or variable name
- WAITFOR <N>

AssignVarReg

Define variable VAR and assign it to a sequence register number (REGNO). VAR must contain alphanumeric characters (a-z, A-Z) and may contain numbers (0-9). VAR cannot start with a number. Max length of VAR: 20 characters.

Max number of variables (including those assigned using AssignVar): 32

0 <= REGNO <= 11

The register (REGNO) can have its value modified using the JSON API command «Sequence/SetRegister».

Parameters:

- VAR - name of the variable (0-11 are valid registers)
- REGNO <N> - register number
- WAITFOR <N>

Alias

Create an alias: NAME = VALUE. Commands that take numerical values as parameters can replace the numerical value with NAME, making it possible to make a change in only one place. It can also make the sequence program more readable.

NAME must contain alphanumeric chars (a-z, A-Z) and may contain numbers (0-9). NAME cannot start with a number.

Max length of NAME: 20 characters.

Max number of aliases: 64

Parameters:

- NAME - name of the alias
- VALUE <N> - value of the alias

Note: All Math commands work with signed numbers. Be careful of rollover. The sequencer CPU is a 16-bit processor.

Add

Result: VAR_A = VAR_A + VALUE or VAR_A = VAR_A + VAR_B VAR_A must be defined. If the 2nd parameter is a variable, VAR_B must be defined.

This is a signed operation, meaning valid range is: -32768 -> 32768

Parameters:

- VAR_A <V> - variable to add to
- VALUE/VAR_B <N/V> - value or variable to add to VAR_A
- WAITFOR <N>

Mult

Result is: VAR_A = VAR_B * VAR_C

Parameters:

- VAR_A <V>
- VAR_B <V>
- VAR_B <V>
- WAITFOR <N>

And

Result is: VAR_A = VAR_B & VAR_C

Parameters:

- VAR_A <V>
- VAR_B <V>
- VAR_B <V>
- WAITFOR <N>

Or

Result is: VAR_A = VAR_B | VAR_C

Parameters:

- VAR_A <V>
- VAR_B <V>
- VAR_B <V>
- WAITFOR <N>

Xor

Result is: VAR_A = VAR_B ^ VAR_C

Parameters:

- VAR_A <V>
- VAR_B <V>
- VAR_B <V>
- WAITFOR <N>

Not

Result is: VAR_A =~ AR_B

Parameters:

- VAR_A <V>
- VAR_B <V>
- WAITFOR <N>

ShiftLeft

Shift the variable VAR, NUMBITS bits to the left.

Parameters:

- VAR <V>
- NUMBITS <N> - number of bits to shift, max 3.
- WAITFOR <N>

ShiftRight

Shift the variable VAR, NUMBITS bits to the right.

Parameters:

- VAR <V>
- NUMBITS <N> - number of bits to shift, max 3.
- WAITFOR <N>

JumpIf

Jump to a specific (labeled) location in the sequence if a condition is met. The condition is a comparison of two variables; VAR_A and VAR_B based on the OPERATOR.

OPERATOR must be either a «less-than» sign (<) or a «greater-than» sign (>).

If the condition is true, then jump to location LABEL, otherwise go to the next line in the sequence file.

Note: The last line in a sequence file must be a «Jump» or «JumpIf» command.

LABEL must be defined earlier with the Label command, while the variables VAR_A and VAR_B must be assigned earlier with the AssignVar command.

Parameters:

- VAR_A <V>
- OPERATOR - < or >
- VAR_B <V>
- LABEL - name of the label to jump to if true
- WAITFOR <N>

Jump

Jump to a specific (labeled) location in the sequence.

Note: The last line in a sequence file must be a «Jump» or «JumpIf» command.

LABEL must be defined earlier with the Label command.

Parameters:

- LABEL - name of the label to jump to
- WAITFOR <N>

Trig

MODE can have the following values: 0 = Positive edge triggering 1 = Negative edge triggering 2 = High level trigger 3 = Low level trigger

SOURCE is given as a 5-bit bitmask. Each source has its defined bit position and several sources can be combined: Bit 0 (binary 00001)= Electrical sync in Bit 1 (binary 00010)= Optical sync in Bit 2 (binary 00100)= Internal sync Bit 3 (binary 01000)= Software sync Bit 4 (binary 10000) = Delayed sync * (see chapter "Triggering options")

Examples: Electrical sync in: SOURCE = 00001 (bin)= 1 (dec). Electrical and internal sync in: SOURCE = 00101 (bin) = 5 (dec).

0 is wait for sync (no timeout)

Note: The delayed sync source is set using JSON API command «Sequence/SetTrigSource». The actual delay is set using the JSON API command «Sequence/SetTrigDelay». Requires Positive edge triggering (MODE = 0)

Parameters:

- MODE <N> - the mode to use, 0-3 is valid
- SOURCE <N> - the source bitmask
- TIMEOUT <N> - 0 is no timeout, otherwise wait for x ticks.

Wait

VALUE is composed of BASE and WAITFOR

Base tells whether the wait is in terms of TICKS or Milliseconds 0 => the wait is in terms of ticks 1 => the wait is in terms of millisconds

Parameters:

- VALUE <N> - VALUE = base*2^22 + waitfor/var

LightSetWord

Set the «Light-control» output pins according to VALUE.

VALUE is 8 bits. Bit(0) => '1' : red strobe on; '0': red strobe off Bit(1) => '1' : blue strobe on; '0': blue strobe off Bit(2) => '1' : green strobe (frame_sync) on; '0': green strobe (frame_sync) off Bit(3:7) => Unused, reserved for future use. Should be set to '0'.

Parameters:

- VALUE <N> - bitmask
- WAITFOR <N>

LightPulseWord

Set the «Light-control» output pins according to VALUE. The value will be active for a DURATION. The DURATION must be set using the JSON API command «LightSource/SetLightPulseDuration».

VALUE is 8 bits. Bit(0) => '1' : red strobe on; '0': no action for red strobe Bit(1) => '1' : blue strobe on; '0': no action for blue strobe Bit(2) => '1' : green strobe (frame_sync) on; '0': no action for green strobe Bit(3:7) => Unused, reserved for future use. Should be set to '0'.

Parameters:

- VALUE <N> - bitmask
- WAITFOR <N>

OutputSetBit

Set the «Output-control» pins where VALUE is 1. Leave the bit positions where VALUE is '0' unchanged.

VALUE is 16 bits. Bit(3, 2 and 0) => Control GPIO(3, 2 and 0). Note: Bit 1 shows LED-on signal. Bit(15:4) => Unused, reserved for future use. Should be set to '0'.

Parameters:

- VALUE <N> - bitmask
- WAITFOR <N>

OutputClearBit

Clear (set to 0) the «Output-control» pins where VALUE is 1. Leave the bit positions where VALUE is '0' unchanged.

VALUE is 16 bits Bit(3, 2 and 0) => Control GPIO(3, 2 and 0). Note: Bit 1 shows LED-on signal Bit(15:4) => Unused, reserved for future use. Should be set to '0'.

Parameters:

- VALUE <N> - bitmask
- WAITFOR <N>

OutputSetWord

Set the «Output-control» pins according to VALUE.

VALUE is 16 bits Bit(3, 2 and 0) => Control gpio(3, 2 and 0). Note: Bit 1 shows LED-on signal Bit(15:4) => Unused, reserved for future use. Should be set to '0'.

Parameters:

- VALUE <N> - bitmask
- WAITFOR <N>

SetMaskImage

Choose an image to use as a mask. Inum 0 will disable masking (but will often need a sequencer reset after).

This command must use a WAIT_FOR value > 75 for all settings to propagate through the system. Sequencer reset will not disable masking.

Due to RAM bandwidth limitation the load row is increased from 40 ns pr row to approximately 46.3 ns pr row (average). E.g. a full image load (1080 rows) must have a WAIT_FOR>=200 (173 without mask image)

Parameters:

- INUM <N> : Inum to use as overlay mask.
- WAITFOR <N>

6.4 Timing Constraints

It takes 40 ns (8 clks a 200 Mhz) to load one row, TICK duration is 250 ns.

Before a new load or reset can be started the row load must be completed. The row load is completed in 40 ns*(number of rows)

	Load	Reset
Load row	LL1 >= (rows*40/250) ticks	LR1 => (row*40/250) ticks
Load group	LL2 >= 12 ticks (72 rows*40/250) [For 1080 DMD]	LR2 => 12 ticks (72 rows*40/250) [For 1080 DMD]
Reset	RL >= 50 ticks (12.5 us)*	RR >= 50 ticks (12.5 us)*
Clear single	CL => 1 tick	CR => 1 tick
Load row	GL => 16 ticks	GR => 16 ticks

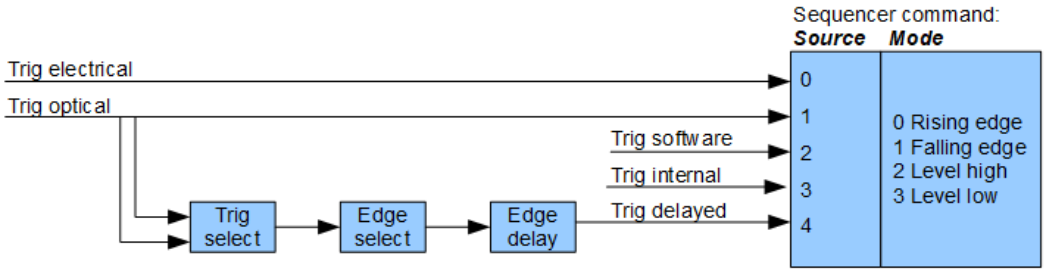
Load to load can be executed back to back as long as the number of rows that is loaded is less than 7 (250/40=6.25 rows can be loaded pr tics). If 1080 rows are loaded then the next load can not start before 173 ticks later (1080*40/250 = 172.8). If MASK image is enabled then the average load time is increased from 40 ns pr row to 46.3 ns pr row.

The same applies for LOAD to RESET. The load must be completed before RESET is executed.

While RST_ACTIVE is high (4.5 us), and for 8us after, the data for the block(s) being reset should not be changed to allow for the settling required for the mirrors to become stable. It is possible to load other blocks while the previously reset block is settling.

6.5 Triggering Options

There are 4 trigger sources plus one which is a delayed version of either the electrical or the optical trigger. Each of these triggers can be used by the sequencer by setting the appropriate source mask and mode. (see «Trig» in Table of commands).



The fifth trig source is a delayed version of either the electrical or the optical trig source. Either positive or negative edge is used and this edge can be delayed up to one pixelheight in steps of 1/256 pixelheight (assuming the selected trig signal is asserted once every pixelheight)

The trig source selection, the edge selection and the delay is controlled by system registers.

The sequencer trig command can select any of the 5 trig sources together with preferred mode.

Only the rising edge mode gives any sense to the fifth source since the signal processing produce a single pulse of either the positive or the negative edge of the optical or electrical trig input.

CHAPTER

SEVEN

WORKFLOW - SCROLLING

In this section we will look at a typical workflow for producing a scrolling image for a moving photohead. This section ONLY applies to LRS-MCX HD V2, LRS-MCX-WQ and LLS products.

7.1 Create a Suitable Bitmap

First we need to get the data to be imaged ready.

LAMA accept image input as:

- BMP (Microsoft BMP style) - Max 4GB image size due to header restriction
- Binary file
- Visitech RLE encoded file

For BMP the CRC checksum for the image is generated realtime by LAMA (in Luxapp). This comes with a small performance cost, as it takes a little bit of time to calculate the CRC on the fly.

As a valid example please look in the sub-directory “example/stripe-files/1920x20000-1bit-inverted.bmp”:

Example stripe

This bitmap is 1920 pixels wide, because we are targetting a HD DMD system. It is 20000 lines long. Notice that we have a "fade-in" dark area on start and end of stripe, since we need to scroll into the first line and out of the last line to ensure all pixels in the image get equal amount of light energy.

7.2 Determine Triggers Needed

For this workflow we are going to move the image on the DMD by 1 line for each light pulse given (trigger). Often refered to as ScrollStep 1. Typically we use a motion stage to produce one trigger each time we have moved the photohead by one pixel in the physical domain.

If we move the photohead by two pixels we would use a "scroll step" of 2 to syncronize motion and the scrolling image.

So we need to produce 20.000 triggers to completely scroll over the example stripe. For scroll step 2 the needed triggers would be 10.000.

7.3 Transfer Data

We assume that we have already added and initialized the photohead in LAMA at this point. Otherwise read the *Get Started* chapter.

To transfer the image we use the API function "Imaging/*LoadStripe*".

Arguments:

- stripe_file = <path to example bitmap file>
- base_inum = 0
- ph_no = 0 (or the correct photohead indentifier)
- rows = 20000
- image_type = "bmp"

Image transfer can take some time to complete if large files are transmitted. So its a good idea to start of this process before doing other things. LAMA works asynchronously so we can save time and complete other tasks while the data is being transferred to the photohead.

7.4 Program the Sequencer

The next thing we want to do is program the sequencer with our required sequence. If you do not know what the sequencer is or does please read the chapter *Sequencer*.

We only need a simple sequence program for our scrolling.

```
#-------------------------------------------------------------
#Command        Parameters                                Waitfor
#-------------------------------------------------------------
AssignVarReg    TotalRows   7                           1
AssignVar       LoopCount   0                           1
AssignVar       MemStartRow 0                           1
AssignVar       Inum        0                           1
```

(continues on next page)

(continued from previous page)

```
AssignVar       DmdStartRow 0                           1
AssignVar       NoOfRows    1080                        1
#
#-------------------------------------------------------------
# Scroll forward
Label           scrollforward                           1
ResetGlobal                                             40  <== Reset DMD
LightPulseWord  15                                      1   <== Set light pulse
LoadRow         DmdStartRow NoOfRows Inum MemStartRow   200 <== Load 1080 rows/lines
Trig            0 4                                     0   <== Image loaded, wait for␣
↪trigger
Add             MemStartRow 1                           1   <== Add one to address␣
↪(scroll step 1)
Add             LoopCount 1                             1   <== Loop to amount of rows
JumpIf          LoopCount < TotalRows scrollforward     1
#
#-------------------------------------------------------------
# Stop
Label           stop                                    1   <== Sequencer CPU idling␣
↪after finish
Jump            stop                                    1   <== Last line must always be␣
↪Jump.
```

We load this sequence by submitting it as a string to the API function "Sequence/*Load*".

Observe we set the TotalRows variable value based on contents of sequencer register number 7. We use the API function "Sequence/*SetRegister*" to do this. We set it with the total rows for the example bitmap (20.000).

Notice that we use "Trig 0 4". This means we want to use the internal frequency generator for the triggers. This is a generator that runs constantly to provide a given frequenzy. If we want to use external optical trigger input we would use "Trig 0 2".

7.5 Set Light Output

We set the light source drivers amplitude level to the value we need using the API function "LightSource/*SetAmplitude*".

7.6 Start Sequencer

Everything should now be ready. We start the sequencer with the API function “Sequence/*StartStop*”.

7.7 Verify

We need to verify that the imaging was completed. We do this either via polling the API function “Sequence/*GetTriggerInfo*” and looking at the returned value for “sequencer_pulses” or waiting for “Sequence/*WaitForTriggers*” to return.

7.8 In Code

We provide a Python code example for this workflow in the sub-directory “example/load-stripe.py”.

7.9 Tips

If you can fit two bitmaps into the memory of the photohead you can upload a new bitmap while imaging the previous, so that you do not have to wait while the previous bitmap has been imaged. This is done by dividing the memory into segments using the sequencer memory adressing and the “base_inum” value of Imaging/LoadStripe.

However colliding a load and a imaging processes in memory will cause undefined behaviour.

CHAPTER

EIGHT

PROTECTIONS

There are several protections making sure that photoheads are not damaged during normal operation.

By using the JSON API function System/*GetPhotoheads* any protections that have been triggered can be read out.

To clear any of the protections from the list, use System/*ClearErrors*.

8.1 over_temperature_formatter_board (OTPB)

Luxbeam Formatter Board has reached its designated max temperature. If the temperature gets higher the board may stop to protect itself from damage.

Check that the cooling to the photohead is sufficient.

8.2 over_temperature_light_source_driver (OTPD)

The light source driver has reached its designed max temperature. Light output may be erratic (blinking or pulsing) because of the protection circuit.

Check that the cooling to the photohead is sufficient.

8.3 over_temperature_light_source_diode (OTPLS)

The light source diode (LED) has reached its designated max temperature. Light output may be erratic (blinking or pulsing) because of the protection circuit limiting current to avoid higher temperature.

Check that the cooling to the photohead is sufficient.

8.4 over_current_light_source (OCP)

The light source diode (LED) requires more current (Amp) than acceptable to produce the set amplitude. Please adjust the amplitude value down for the driver.

This may also occur when the light source diode is nearing its end of life. Current draw will go up for a given amount of light, emitting the power as heat rather than light. Check the remaining light source life.

8.5 over_power_light_source (OPP)

The light source diode (LED) requires more power than acceptable to produce the set amplitude. Check for bad connections or short-circuits.

8.6 door_open_switch (DOOR)

The light safety switch has been triggered. Check that the door protection circuit is in a closed state. This protection is in place to avoid accidental exposure to dangerous powerful light emitted from the photohead.

CHAPTER

NINE

API DESCRIPTION

9.1 System

9.1.1 GetVersion

System/GetVersion

Get the version of the running service.

Response:

- **version <string>** - The version of the service.
- **major <number>** - The major version number.
- **minor <number>** - The minor version number.
- **micro <number>** - The micro version number.

9.1.2 AddPhotohead

System/AddPhotohead

Used to add a photohead to the system.

Arguments:

- **ph_no <number>** - Number to use for later addressing the photohead.
- **ip <string>** - IP address of the photohead to add.
- **type <** *ProductID* **>** - Type of product. Valid photohead types found under *ProductID* type.
- **dmd_rotated <bool>** - *OPTIONAL.* Set DMD rotation. If set FALSE dmd axis is parallel to the motion axis. If set TRUE, the dmd axis is 180 degrees rotated to the motion axis. The default is TRUE. Will affect only y-scrolling systems.

```
{
    "module": "System",
    "cmd": {
        "func": "AddPhotohead",
        "args": {
            "ph_no": 0,
            "ip": "192.168.0.10",
            "type": "product-x",
            "dmd_rotated": true
```

(continues on next page)

(continued from previous page)

```
        }
    }
}

{
    "status": "ok"
}
```

9.1.3 RemovePhotohead

System/RemovePhotohead

Used to remove a photohead from the system. All communications started before this command is issued will be allowed to complete before the photohead is removed.

Arguments:

- **ph_no <number>** - Number addressing the photohead.

```
{
    "module": "System",
    "cmd": {
        "func": "RemovePhotohead",
        "args": {
            "ph_no": 0
        }
    }
}

{
    "status": "ok"
}
```

9.1.4 InitPhotohead

System/InitPhotohead

Used to initialize a photohead in the system. Must be run before anything else is done with the photohead. Photoheads inited will be inited automatically the next time the application is started.

Arguments:

- **ph_no <number>** - *OPTIONAL* Photohead number. If no ph_no is given, all photoheads are initialized.
- **start_tray <bool>** - *OPTIONAL.* Start the Luxapp GUI for this. Default TRUE.

```
{
    "module": "System",
    "cmd": {
        "func": "InitPhotohead",
        "args": {
            "ph_no": 0
        }
```

(continues on next page)

(continued from previous page)

```
    }
}

{
    "status": "ok"
}
```

9.1.5 ShutDownPhotohead

`System/ShutDownPhotohead`

Safely shutdown photoheads. This will turn off all photoheads in arguments. They will have to be power cycled to be initialized again.

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.

9.1.6 GetPhotoheads

`System/GetPhotoheads`

Get all photoheads managed and their status.

Arguments: None

Response:

- **photoheads <vector<objects>>**:
  - **ph_no <number>** - Photoheads assigned number.
  - **type <string>** - Photohead type.
  - **version <string>** - Firmware version.
  - **ip <string>** - IP of photohead.
  - **is_inited <bool>** - Has the photohead been initialized.
  - **is_initing <bool>** - Is the photohead in the process of initializing.
  - **is_alive <bool>** - Is the photohead still alive (connected).
  - **dmd_width <number>** - Only present if DMD is connected. The width in pixels.
  - **dmd_height <number>** - Only present if DMD is connected. The height in pixels.
  - **dmd_type <number>** - Only present if DMD is connected. The DMD type connected.
  - **dmd_rotated <bool>** - Is the DMD rotated. Only present for products capable of scroling.
  - **print_enable <bool>** - Is the photohead enabled for printing. Only present for products capable of scroling.
  - **is_up_to_date <bool>** - Is the photohead firmware up to date.
  - **ppc_on <bool>** - Is Pixel Power Control on.
  - **has_programmable_sequencer <bool>** - Does the photohead have a programmable sequencer.

- **temperatures <object>** - Only present if sensors is installed. If temperature is 0C then sensor is not connected.
  - **mainboard <number>** - Mainboard temperature.
  - **mainboard_second <number>** - Second mainboard temperature.
  - **fpga <number>** - FPGA temperature.
  - **regulator_1 <number>** - Regulator 1 (heater) temperature.
  - **regulator_2 <number>** - Regulator 2 (heater) temperature.
- **upgrade_status <object>** - Only present if firmware upgrade is in progress.
  - **phase <string>** - The current phase of the upgrade. Can be "erasing", "writing", "verifying" or "restart".
  - **progress <number>** - A number between 0 and 100 indicating the progress of the upgrade.
  - **message <string>** - A message describing the current phase of the upgrade.
  - **section <string>** - The current section of the upgrade. Some devices have multiple sections. If so they are indicated here. E.g "bootloader", "application"
- **protections <object>** - Which protections have been triggered.
  - **over_temperature_formatter_board <bool>** - Formatter board over temperature state triggered.
  - **over_temperature_light_source_driver <bool>** - Light Source driver over temperature state triggered.
  - **over_temperature_light_source_diode <bool>** - Light source diode over temperature state triggered.
  - **over_current_light_source <bool>** - Light source has reached the electrical current limit.
  - **over_power_light_source <bool>** - Light source has reached power limit.
  - **door_open_switch <bool>** - The light safety switch has been triggered.

9.1.7 OpenPhotoheadControlPanel

`System/OpenPhotoheadControlPanel`

Opens the graphical user interface for this photohead.

Arguments:

- **ph_no <number>** - Photohead to interact with.

9.1.8 SetPhotoheadLensFactor

`System/SetPhotoheadLensFactor`

Set lens factor for this photohead.

Arguments:

- **ph_no <number>** - Photohead to interact with.
- **lens_factor <float>** - Lens factor to set.

9.1.9 GetPhotoheadLensFactor

System/GetPhotoheadLensFactor

Get lens factor for this photohead.

Arguments:

- **ph_no <number>** - Photohead to interact with.

Response:

- **lens_factor <float>** - Lens factor for this photohead.

9.1.10 Ping

System/Ping

Can be used to verify that the service is up and running and ready for commands.

Arguments:

- ph_no - Photohead number to ping.

```
{"module": "System", "cmd":
    {"func": "Ping", "args": {"ph_no": 0}}}

{"status": "ok"}
```

9.1.11 ClearErrors

System/ClearErrors

Clear all protection errors triggered in photoheads.

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.

9.1.12 UpgradePhotoheadFirmware

System/UpgradePhotoheadFirmware

Upgrade the firmware of a photohead (if a new firmware is available).

This is a long running operation that can not be interrupted. The photohead must be restarted after the firmware upgrade is completed.

Note: When the upgrade is in progress, *GetPhotoheads* will also contain the firmware upgrade status.

Arguments:

- **ph_no <number>** - The photohead to upgrade.

Response:

- **status <string>** - "working" or "ok". If "working" the upgrade is still in progress.
- **ret <object>** - The return object contains information about the progress of the upgrade.
- **message <string>** - A message describing the current phase of the upgrade.

- **progress <number>** - A number between 0 and 100 indicating the progress of the upgrade.
- **phase <string>** - The current phase of the upgrade. Can be “erasing”, “writing”, “verifying” or “restart”.

```
{
    "module": "System",
    "cmd": {
        "func": "UpgradePhotoheadFirmware",
        "args": {
            "ph_no": 0
        }
    }
}

{
    "status": "working",
    "ret" : {
        "message": "Firmware upgrade started"
        "progress": 0
        "phase": "erasing"
    }
}

{
    "status": "working",
    "ret" : {
        "message": "Erasing flash"
        "progress": 50
        "phase": "erasing"
    }
}

{
    "status": "working",
    "ret" : {
        "message": "Writing flash"
        "progress": 50
        "phase": "writing"
    }
}

{
    "status": "working",
    "ret" : {
        "message": "Verifying written content"
        "progress": 50
        "phase": "verifying"
    }
}

 {
    "status": "working",
    "ret" : {
        "message": "Restart photohead now"
```

(continues on next page)

(continued from previous page)

```
        "progress": 0
        "phase": "restart"
    }
}

{
    "status": "ok",
    "ret" : {
        "message": "Firmware upgrade completed"
        "progress": 100
        "phase": "done"
    }
}
```

9.1.13 GetTemperatureRegulator

System/GetTemperatureRegulator

Get the target temperature of temperature regulators in photohead(s).

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.

Response:

- **photoheads <array>** - Array of photoheads with the following information:
  - **regulators <array>** - Array with all regulators for this photohead.
    - **regulator_no <number>** - The regulator number.
    - **enabled <bool>** - Is the regulator enabled.
    - **set_point <number>** - The temperature set-point in degrees Celsius.
    - **current_temperature <number>** - The current temperature in degrees Celsius.

9.1.14 SetTemperatureRegulator

System/SetTemperatureRegulator

Set the target temperature of temperature regulators in photohead(s).

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.
- **regulator_no <number>** - The regulator to set the temperature for. Typically one or two are installed.
- **enabled <bool>** - Enable or disable the regulator.
- **set_point <number>** - A temperature set-point in degrees Celsius.

9.1.15 SetPhotoheadMacAddress

`System/SetPhotoheadMacAddress`

Set the MAC address of a photohead. Stored permanentely across reboots of system. The new MAC address is taken into use immediately. Remember that you may need to flush ARP cache of the system you are using to communicate with the photohead.

Arguments:

- **ph_no <number>** - Photohead to interact with.
- **mac <string>** - A valid MAC address. In format "00:11:22:33:44:55".

9.1.16 SetPhotoheadIpAddress

`System/SetPhotoheadIpAddress`

Set the IP address of a photohead. Stored permanentely across reboots of system. The new IP address is taken into use immediately. Remember that you may need to adjust the IP address of the systeam you are using to communicate with the photohead.

Arguments:

- **ph_no <number>** - Photohead to interact with.
- **ip <string>** - A valid IPv4 address.

9.2 Sequence

9.2.1 StartStop

`Sequence/StartStop`

Start or stop the sequencer on all photoheads specified.

Arguments:

- toggle (boolean) - Start or stop the sequencer.
- photoheads <vector<number>> - Photoheads to interact with.

```
{"module": "Sequence", "cmd":
    {"func": "StartStop", "args": {
        "photoheads": [0,1,2,3],
        "toggle": true
    }}}

{"status": "ok"}
```

9.2.2 Select

Sequence/Select

Select internal/integrated test sequence or external user-loaded sequence.

Arguments:

- sequence (string) - "internal" or "external"
- photoheads <vector<number>> - Photoheads to interact with.

```
{"module": "Sequence", "cmd":
    {"func": "Select", "args": {
        "sequence": "internal"
        "photoheads": [0,1,2,3]
    }}}

{"status": "ok"}
```

9.2.3 GetSelected

Sequence/GetSelected

Get the selected sequence, internal or external.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.

Response:

- selected (array of string) - Selected sequence for each photohead.

9.2.4 Load

Sequence/Load

Load a sequence. See own documentation for writing sequencer files. This command will automatically stop and reset the sequencer before loading the sequence to all the photoheads specified.

Arguments:

- sequence (string) - The sequence in an escaped string form.
- photoheads <vector<number>> - Photoheads to interact with.

```
{"module": "Sequence", "cmd":
    {"func": "Load", "args": {
        "sequence": "LoadRow and so on....."
        "photoheads": [0,1,2,3]
    }}}

{"status": "ok"}
```

9.2.5 Reset

Sequence/Reset

Reset the sequencer. The sequence will start from the beginning. This command will not stop the sequencer.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.

```
{"module": "Sequence", "cmd":
    {"func": "Reset", "args": {
        "photoheads": [0,1,2,3]
    }}}

{"status": "ok"}
```

9.2.6 SetRegister

Sequence/SetRegister

Set a sequencer register, that can be accessed from the sequence file later. There is a total of 12 sequencer registers that can be used.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- register (number) - The sequencer register to set, (0-11 valid).
- value (number) - The value to set. Signed 16-bit value max.

```
{"module": "Sequence", "cmd":
    {"func": "SetRegister", "args": {
        "photoheads": [0,1,2,3],
        "register": 1,
        "value" 4000
    }}}

{"status": "ok"}
```

9.2.7 SetInternalFrequency

Sequence/SetInternalFrequency

Set the frequency of the internal trigger generator.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- hz (number) - The frequency in Hz to set.

```
{"module": "Sequence", "cmd":
    {"func": "SetInternalFrequency", "args": {
        "photoheads": [0,1,2,3],
        "hz": 15000
```

(continues on next page)

(continued from previous page)

```
    }}}

{"status": "ok"}
```

9.2.8 SetTriggerSource

Sequence/SetTriggerSource

The trigger source to be used as input trigger to the sequencer.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- source (string) - “optical” or “electrical”.
- edge (string) - “positive” or “negative”.

9.2.9 SetTriggerDelay

Sequence/SetTriggerDelay

This manipulates the delay of the «Delayed Trigger Input» source, that can be selected from the sequencer.

Delay the active trigger pulse by a number of 1/256 parts of a trigger period.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- delay (number) - Range 0-255, 0 is 0 delay.

9.2.10 SetLightPulseDuration

Sequence/SetLightPulseDuration

This function manipulates the duration of the light pulse that is applied when a «LightPulseWord» command is used in the sequence file.

For products without a programmeable sequencer, it is used by the internal sequencer to set the light exposure time.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- duration (number) - The duration the light pulse should be on. Unit: 250 ns. (Hence duration 1 = 250 ns, duration 65535 = 16.37375 ms)

9.2.11 SetMirrorShake

`Sequence/SetMirrorShake`

Switch the DMD mirrors from on to off position continuously. This action will preserve the longevity of the DMD and avoid e.g stuck mirrors.

Enable this as often as possible, e.g between exposures, or when the photohead is idling to increase the life span of the DMD.

Note: the light source drivers will be turned off when mirror shake is enabled, and the sequencer will load a special sequence. Before next exposure mirror shake will have to be disabled. When mirror shake is enabled the sequencer will be reset, so any sequencer registers will also be reset!

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- toggle (boolean) - true or false to enable or disable.

9.2.12 GetRunningStatus

`Sequence/GetRunningStatus`

Get the running status of the sequencer (on/off).

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.

Response:

- running (array of booleans) - Status for each photohead requested.

9.2.13 GetTriggerInfo

`Sequence/GetTriggerInfo`

Get count of how many triggers has been received and executed by the sequencer.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.

Response

•array of dictionaries:

- in_pulses_electrical - Number of triggers received on electrical interface.
- in_pulses_optical - Number of triggers received on optical interface.
- min_period_electrical - Minimum period between two trigger pulses on electrical trigger input pin (since last read). Unit: 10-8 sec
- min_period_optical - Minimum period between two trigger pulses on optical trigger input pin (since last read). Unit: 10-8 sec
- min_period_number - Number of the trigger that had the shortest period.
- sequencer_pulses - Number of triggers executed by sequencer.

9.2.14 WaitForTriggers

`Sequence/WaitForTriggers`

Wait for the photoheads to receive the expected number of electrical, optical or internal trigger pulses.

This function can be stopped while running, with a stop request described in the protocol description.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- triggers (number) - The number of trigger pulses to wait for.
- limit_ms (number) - A time limit for all the triggers to arrive, given in milliseconds.

9.2.15 TimedStaticExposure

`Sequence/TimedStaticExposure`

Run sequencer for a set amount of time, with a static image, with LED turned on.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- exposure_time (number) - Exposure time given in milliseconds.
- inum (number) - The inum to use for the exposure.

9.2.16 SetImageOrientation

`Sequence/SetImageOrientation`

Flip the image orientation on the DMD.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- flip_horizontal (bool) - Flip the image horizontally.
- flip_vertical (bool) - Flip the image vertically.

9.2.17 GetImageOrientation

`Sequence/GetImageOrientation`

Get the current image orientation on the DMD.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.

Response: * array of dictonaries:

- flip_horizontal - Flip the image horizontally.
- flip_vertical - Flip the image vertically.

9.2.18 SetDMDPark

`Sequence/SetDMDPark`

When a DMD is idle and not actively projecting data use this to park the mirrors to assist in maximizing DMD lifetime. Use Sequence/SetMirrorShake during shorter waiting periods of DMD non-operation. For waiting periods longer than 2 minutes, park the DMD.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.
- park (bool) - Park or unpark the DMD, (TRUE) means park.

9.2.19 GetDMDPark

`Sequence/GetDMDPark`

Get the park state for DMD.

Arguments:

- photoheads <vector<number>> - Photoheads to interact with.

Response:

- array of booleans - Park state for each photohead requested.

9.2.20 SendTriggerPulse

`Sequence/SendTriggerPulse`

Send a trigger pulse to the photoheads specified. Can NOT be used to in a syncronized way to multiple photoheads.

Arguments:

- **ph_no <number>** - Photohead to interact with.

9.3 LightSource

9.3.1 SetAmplitude

`LightSource/SetAmplitude`

Set the light amplitude for each light source driver.

Arguments:

- **photoheads <vector<numbers>>** - Photoheads to interact with.
- **amplitudes <vector of objects>**
  - **driver_no <number>** - Driver number to interact with.
  - **value <number>** - The amplitude value to set. Range 0-4096.

```
{
    "module": "LightSource",
    "cmd": {
        "func": "SetAmplitude",
        "args": {
            "photoheads": [0, 1],
            "amplitudes": [
                {"driver_no": 0, "value": 1000},
                {"driver_no": 1, "value": 1000}
            ]
        }
    }
}

 {"status": "ok"}
```

9.3.2 GetStatus

`LightSource/GetStatus`

Get status and current configuration of a light source driver.

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.

Response:

- **(vector of objects, one for each photohead requested)**
  - **(vector of objects, one for each driver present in photohead)**
    - **amplitude_set <number>** - Current light amplitude set.
    - **ocp_set <number>** - Over-current protection value set (in mA).
    - **ocp_current_max_allowed <number>** - Max Over-current protection allowed (in mA).
    - **regulation_mode_set <string>** - Current regulation mode in use (LIGHT, CURRENT or COMBINED)
    - **board_temperature <number>** - Current light source driver temperature in Celsius.
    - **led_temperature <number>** - Current light source LED temperature in Celsius.
    - **driver_type <string>** - Type of light source driver connected.
    - **light_feedback <number>** - Light feedback sensor value.
    - **current_feedback_sensor <vector<numbers>>** - Current feedback (in mA) for each loop on driver.
    - **driver_no <number>** - Light source driver number. Starts from 0.
    - **door_open_protection <bool>**
    - **driver_over_temperature_protection <bool>**
    - **led_over_temperature_protection <bool>**
    - **duty_cycle_protection <bool>**
    - **over_current_protection <bool>**

        * **over_power_protection <bool>**
        * **on <bool>** - Is the driver enabled.
        * **build <object>** - Build date and time of driver firmware.
            - **year <number>** - Year of driver build.
            - **month <number>** - Month of driver build.
            - **day <number>** - Day of driver build.
            - **hour <number>** - Hour of driver build.
            - **minute <number>** - Minute of driver build.
            - **second <number>** - Second of driver build.

9.3.3 StartStop

`LightSource/StartStop`

Enable or disable a light source. Note: Driver will not power light source unless there is an active trigger source from the sequencer.

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.
- **no <vector<number>>** - Light source drivers to interact with.
- **enable <bool>** - Turn driver on or off. True = Light on. NB: Argument name “toggle”, deprecated.

```
{
    "module": "LightSource",
    "cmd": {
        "func": "StartStop",
        "args": {
            "photoheads": [0, 1],
            "no": [0, 1, 2, 3],
            "enable": true
        }
    }
}

{
    "status": "ok"
}
```

9.3.4 SetOCPLimit

`LightSource/SetOCPLimit`

Set the OCP limits for a set of photo heads and light sources

Arguments:

- **photoheads <vector<number>>** - The photo heads to set OCP limit to.
- **ocp_limits (vector of the following items)**

  - **driver_no <number>** - The light source number to set OCP limit to.
  - **value <number>** - The OCP limit value.

```
{
    "module": "LightSource",
    "cmd": {
        "func": "SetOCPLimit",
        "args": {
            "photoheads": [0, 1]
            "ocp_limits": [
                {
                    "driver_no": 0,
                    "value": 1000
                },
                {
                    "driver_no": 1,
                    "value": 1000
                }
            ]
        }
    }
}

{
    "status": "ok"
}
```

9.3.5 SetMaxCurrent

LightSource/SetMaxCurrent

Set the max current for a set of photo heads and light sources. This limit will result in a softer current limit than the OCP limit. The OCP limit is a hard limit that will turn off the light source if exceeded. The max current limit will instead reduce the light output of the light source if exceeded.

Note: Only support by a few drivers:

- LASER IR V1

Arguments:

- **photoheads <vector<number>>** - The photo heads to set OCP limit to.
- **max_currents (vector of the following items)**
  - **driver_no <number>** - The light source number to set OCP limit to.
  - **value <number>** - The OCP limit value.

```
{
    "module": "LightSource",
    "cmd": {
        "func": "SetMaxCurrent",
        "args": {
            "photoheads": [0, 1]
            "max_currents": [
```

(continues on next page)

(continued from previous page)

```
                {
                    "driver_no": 0,
                    "value": 1000
                },
                {
                    "driver_no": 1,
                    "value": 1000
                }
            ]
        }
    }
}

{
    "status": "ok"
}
```

9.3.6 SetRegulationMode

LightSource/SetRegulationMode

Set the regulation mode of a light source driver. Regulate either by the light source feedback sensor or by the current feedback sensor. Light regulation is always recommended as this will make sure that the light output is always the same even if temperature, age or other factors change the light output of the LED over time. Current mode may be used for special applications where very small light output is required.

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.
- **modes (list)**
  - **driver_no <number>** - Driver number to interact with.
  - **mode <string>** - Mode to use. Can be "LIGHT" or "CURRENT".

```
{"module": "LightSource", "cmd": {
    "func": "SetRegulationMode", "args": {
        "photoheads": [0,1],
        "modes": [
            {"driver_no": 0, "mode": "LIGHT"}
            {"driver_no": 1, "mode": "LIGHT"} ]
        }
    }
}

{"status": "ok"}
```

9.3.7 SetCalibrationTable

`LightSource/SetCalibrationTable`

The main goal of this feature is to provide a linear power delivery. Furthermore we will also provide a custom power value per amplitude point. E.g each amplitude increment is worth 5mw.

40 points were selected because it corresponds good with the maximum amplitude value of 0-4095 (12 bit).

Given 40 points across 0-4095 it provides enough points for the whole range with steps of 100 amplitude points. Point 0 applies to 0-99, Point 1 applies to 100-199 and so on.

Calibration points are divisors applied to the amplitude. The amplitude is multiplied by 1000 before being divided by the divisor defined in the calibration table. So, for a divisor of 1100 an amplitude of 1000 will be about 909: (amplitude*1000/divisor = 1000*1000/1100 ~ 909), the remainder of the calculation is not used, the math is integer-based. After setting the table it is stored permanently in the main board and loaded on boot. The table can be set whenever the user wishes to do so.

To default the table the user set all points to 1000. The table is stored on the mainboard of the photohead, and loaded again on startup.

Arguments:

- **ph_no (number)** - Photohead to interact with.
- **driver_no <number>** - The light source driver to interact with.
- **calibration_points <vector of numbers>** - The calibration points to set.

9.4 Imaging

9.4.1 LoadTestImage

`Imaging/LoadTestImage`

Load a built-in test image.

Arguments:

- **image_no <number>** - Test image number to load. 0-4 valid.
- **start_sequencer <bool>** - Immediately start the sequencer.
- **photoheads <vector<number>>** - Photoheads to interact with.
- **mirror_shake_off <bool>** - *OPTIONAL.* Turn mirror shake off.

9.4.2 LoadImage

`Imaging/LoadImage`

Load an image which corresponds to one DMD height (and width) into a given image number offset.

This function requires BSON to be used if a bitmap is sent directly and not path.

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.
- **inum <number>** - Image number offset to load image.
- **image_type <string>** - “bmp” or “binary”.

- **image <byte array>** - *OPTIONAL*
- **image_path <string>** - *OPTIONAL.* Path to image file.

9.4.3 LoadStripe

`Imaging/LoadStripe`

Load a stripe file.

Use the function Sequence/SetInternalFrequency to set the trigger rate for internal trigger mode.

For photoheads with a programmeable sequencer the arguments "trigger_mode" and "direction" are ignored. As triggering and scroll direction is controlled by the use of the sequencer language.

WARNING: Make sure that MTU have been adjusted to 9000B before trying to use this function (jumbo packets required).

Arguments:

- stripe_file (string) - Fully qualified file path on the local system.
- base_inum (number) - Offset in memory to load stripe.
- ph_no (number) - Photohead to interact with.
- rows (number) - Number of lines to load from stripe file.
- image_type (string) - "bmp" or "bin".
- trigger_mode (string) - *OPTIONAL* "external", "internal". If "external" is selected, the photohead will wait for an external trigger to load the next image. If "internal" is selected, the photohead will load images at a fixed rate.
- direction (string) - *OPTIONAL* "forward" or "reverse". Scroll direction.

Response while working:

- status (string) - "working"
- rows_loaded - (number) How many rows have been loaded so far. Sent every 300ms.
- crc_check - (string) "working" or "ok"

Response when done:

- status (string) - "ok"
- rows_loaded - (number) How many rows were loaded in total.
- crc_check - (string) "ok"

9.4.4 LoadPixelPowerControlMask

`Imaging/LoadPixelPowerControlMask`

Load mask for Pixel Power Control (PPC). Photoheads require different formats for the mask.

This function requires BSON to be used if a bitmap is sent directly and not a file path.

For LRS MCx-4k the mask should be a 64x34 8-bit bitmap. Each pixel in this image represent a 64x64 pixel area on the photohead from upper left to bottom right.

Arguments:

- **ph_no <number>** - Photohead to interact with.
- **save <bool>** - *OPTIONAL* Flag indicating that the PPC table should be stored in non-volatile memory (flash) after loading. If not set the PPC is not saved.
- **mask <byte array>** - *OPTIONAL* Image mask bitmap.
- **mask_path <string>** - *OPTIONAL* Path to mask file.

9.4.5 SetPixelPowerControl

`Imaging/SetPixelPowerControl`

Enable or disable the Pixel Power Control (PPC) for a photohead.

Arguments:

- **photoheads <vector<number>>** - Photoheads to interact with.
- **enable <bool>** - Enable or disable PPC.

9.4.6 GeneratePixelPowerControlImage

`Imaging/GeneratePixelPowerControlImage`

Generate pixel power control (PPC) image from iradiance measurement data. The generated image is usable for y-scrolling purposes. Use ** :ref:LoadPixelPowerControlMask ** to load the generated image.

Arguments:

- **ph_no <number>** - Photohead to generate ppc image for.
- **path <string>** - Path to save the generated PPC image.
- **iradiance_data vector<number>** - The iradiance dataset to generate ppc from. The function expects evenly spaced data points. The data points should cover the full width of the DMD from left to right edge (in that order).
- **keep_out <number>** - *OPTIONAL* The number of pixels to keep out from the top and bottom edges of the image. This is used if only part of the DMD activ during scrolling. The parameter is optional and defaults to 0 (full DMD height used).
- **num_sections <number>** - *OPTIONAL* The number of sections to spread the PPC masking pixels over in the y direction. The parameter is optional and defaults to 36.

9.4.7 ImageSeries

`Imaging/ImageSeries`

Continously load images from a directory.

Directory should contain .bmp files with the same resolution as the photohead and be named in the following format: “Lama-series-0.bmp” “Lama-series-1.bmp” “Lama-series-2.bmp” etc. Load order is determined by the file name.

Use the function Sequence/SetInternalFrequency to set the trigger rate for internal trigger mode. Use the function Sequence/SetLightPulseDuration to set the light exposure time.

If you want to use software trigger mode you can use the function Sequence/SendTriggerPulse to trigger the next image.

WARNING: Make sure that MTU have been adjusted to 9000B before trying to use this function (jumbo packets required).

Arguments:

- ph_no (number) - Photohead to interact with.
- directory (string) - Fully qualified file path on the local system to load images from.
- trigger_mode (string) - "external", "internal" or "software". If "external" is selected, the photohead will wait for an external trigger to load the next image. If "internal" is selected, the photohead will load images at a fixed rate.

Response while working (sent each 300ms):

- status (string) - "working"
- progress (number) - Progress in percent.
- current_image_no (number) - The current image number in the series being displayed.
- stage (string) - "preparing", "enumerating_directory", "imaging", "waiting_for_trigger", "done".

Response when done:

- status (string) - "ok"
- images_displayed - (number) - The number of images displayed.

9.5 AxisControl

9.5.1 AddController

`AxisControl/AddController`

Add an axis controller device to the system.

An axis controller can control XY and Z axes on the same controller or there may be one or more XY controllers and one or more Z controllers in a system.

Arguments:

- **controller_name <string>** - The name given to the controller.
- **type <string>** - The controller type.

  Supported types are: **"sm_mc2"**, **"sm_mc2_emu"** and **"external"**.
- **ip <string>** - - *OPTIONAL.* IP address of the motion controller to be added. May be dropped for emulated controllers.
- **port <number>** - *OPTIONAL.* The port to be used. This argument may be dropped. If argument dropped, port is set to 13827 which is the default port for Sieb&Meyer controller ("sm_mc2").

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "AddController",
        "args": {
            "controller_name": "XY_and_Z",
            "type": "sm_mc2",
```

(continues on next page)

(continued from previous page)

```
            "ip": "10.60.1.35",
            "port": 13827
        }
    }
}

{
    "status": "ok"
}
```

9.5.2 RemoveController

AxisControl/RemoveController

Remove an axis controller from the system. If any focus drives or tables have been assigned to the controller, then an exception will be given, with information about the tables and focus drives using the controller. These must be removed before the controller can be removed.

Arguments:

- **controller_name <string>** - The name of the controller to remove.

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "RemoveController",
        "args": {
            "controller_name": "XY_and_Z"
        }
    }
}

{
    "status": "ok"
}
```

9.5.3 GetControllers

AxisControl/GetControllers

Get a list of controllers in the system. Information on status and number of supported drives is also given.

Arguments: None

Response:

- **controllers <vector<objects>>** - A vector of objects:
  - **controller_name <string>** - The name of the controller.
  - **type <string>** - The controller type.
  - **ip <string>** - IP address of the controller. Will be "0.0.0.0" if not set.
  - **port <number>** - The port that is used.

- **connection_ok <bool>** - TRUE = Connection to controller has been established.
- **firmware_version <string>** - The firmware version of the controller. Empty string is returned if not possible to connect to controller.
- **num_drives <number>** - The maximum number of drives supported by the device.


```
{
    "module": "AxisControl",
    "cmd": {
        "func": "GetControllers"
    }
}

{
    "ret": {
        "controllers": [
            {
                "controller_name": "XY_and_Z",
                "type": "sm_mc2",
                "ip": "10.60.1.35",
                "port": 13827,
                "connection_ok": "true",
                "firmware_version": "3.0.5",
                "num_drives": 7
            },
            {
                "controller_name": "Z_contr",
                "type": "sm_mc2",
                "ip": "10.60.1.40",
                "port": 13827,
                "connection_ok": "true",
                "firmware_version": "3.0.5",
                "num_drives": 7
            }
        ]
    },
    "status": "ok"
}
```


9.5.4 AddTable

`AxisControl/AddTable`

Add a table to the system. There can be several tables in the system, with different named IDs. A table must have at least an Y axis, but X axis is optional. If X axis is not used, the x-part of coordinates will be ignored.

Arguments:

- **table_name <string>** - Unique name identifying the table.
- **controller_name <string>** - The controller to be used for the table. Must be defined in AddController.
- **x_drive_id <number>** - *OPTIONAL.* The motion controller axis ID to be used for X motion. If argument is skipped, no X-axis will be used.
- **y_drive_id <number>** - The motion controller axis ID to be used for Y motion.

- **max_motion_speed <** Speed **>** - The maximum speed that the motion controller will use. This speed will be used when moving in x or moving to for example exposure start position. Maximum printing speed will also be limited by this setting.
- **acceleration_dist <** Length **>** - The y-acceleration distance. When printing, the start point of the motor will be offset by a length of acceleration_dist compared to the exposure start position.
- **table_limits <json object>** - *OPTIONAL.* The min and max limits for the table. If not supplied, there will be no limits for the table.
  - **min_pos <** Coordinate **>** - The minimum position that the table can move to.
  - **max_pos <** Coordinate **>** - The maximum position that the table can move to.
- **parallelogram_adjust <number>** - *OPTIONAL.* A factor used to compensate for inaccurate table (X and Y axis not perpendicular). A factor below 1 will reduce the angle for the Y-motion while a factor above 1 will increase the angle.

  E.g. factor 1.1 will increase the angle by 10%.

  If this argument is not supplied a factor of 1 is used (i.e. no adjustment).
- **flip_x_axis <bool>** - *OPTIONAL.* If TRUE, the table X axis will be flipped. Default is FALSE.
- **flip_y_axis <bool>** - *OPTIONAL.* If TRUE, the table Y axis will be flipped. Default is FALSE.

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "AddTable",
        "args": {
            "table_name": "LeftTable",
            "controller_name": "XY_and_Z",
            "x_drive_id": 0,
            "y_drive_id": 1,
            "max_motion_speed": [200, "mmps"],
            "acceleration_dist": [5, "mm"],
            "table_limits": {
                "min_pos": [-10.0, -30.0],
                "max_pos": [300.0, 500.0]
            },
            "parallelogram_adjust": 1.0
        }
    }
}

{
    "status": "ok"
}
```

9.5.5 RemoveTable

AxisControl/RemoveTable

Remove a table from the system.

Arguments:

- **table_name <string>** - The name of the table to remove.

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "RemoveTable",
        "args": {
            "table_name": "LeftTable"
        }
    }
}

{
    "status": "ok"
}
```

9.5.6 GetTables

AxisControl/GetTables

Get a list of defined tables in the system.

Arguments: None

Response:

- **tables <vector <objects>>** - A vector of tables that are defined.
  - **table_name <string>** - Name of the table.
  - **controller_name <string>** - The controller used for the table.
  - **x_drive_id <number>** - The motion controller axis ID used for X motion. Returns -1 if axis is not in use.
  - **y_drive_id <number>** - The motion controller axis ID used for Y motion.
  - **max_motion_speed <** Speed **>** - The maximum speed that the motion controller will use.
  - **acceleration_dist <** Length **>** - The y-acceleration distance.
  - **table_limits <json object>** - The min and max limits for the table.
    - **min_pos <** Coordinate **>** - The minimum position that the table can move to.
    - **max_pos <** Coordinate **>** - The maximum position that the table can move to.
  - **flip_x_axis <bool>** - Table X axis flip status.
  - **flip_y_axis <bool>** - Table Y axis flip status.
  - **parallelogram_adjust <number>** - Parallelogram adjustment factor.
  - **is_inited <bool>** - TRUE = The table has been successfully calibrated/initialized.

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "GetTables"
    }
}

{
    "ret":  {
        "tables": [
            {
                "table_name": "LeftTable",
                "controller_name": "XY_and_Z",
                "x_drive_id": 0,
                "y_drive_id": 1,
                "max_motion_speed": [200, "mmps"],
                "acceleration_dist": [5, "mm"],
                "table_limits": {
                    "min_pos": [-10.0, -30.0],
                    "max_pos": [300.0, 500.0]
                },
                "flip_x_axis": false,
                "flip_y_axis": false,
                "parallelogram_adjust": 1.0,
                "is_inited": "true"
            }
       ]
    },
    "status": "ok"
}
```

9.5.7 InitializeTable

AxisControl/InitializeTable

This command will initialize axes that are assigned with the given table. This must be done before the table can be used. The initialization process may take a few seconds.

Arguments:

- **table_name <string>** - Unique name identifying the table.

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "InitializeTable",
        "args": {
            "table_name": "LeftTable"
        }
    }
}

{
```

(continues on next page)

(continued from previous page)

```
    "status": "ok"
}
```

9.5.8 MoveTableToPosition

AxisControl/MoveTableToPosition

Move a table to a given position.

Arguments:

- **table_name <string>** - The table to move.
- **speed <** Speed **>** - *OPTIONAL*. The speed to use for the motion.

  If not supplied, the maximum speed defined for the table will be used.
- **target_pos <** Coordinate **>** - The position to move to.

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "MoveTableToPosition",
        "args": {
            "table_name": "LeftTable",
            "speed": [40, "mms"],
            "target_pos": [-20.0, 50.0]
        }
    }
}

{
    "status": "ok"
}
```

9.5.9 GetTablePosition

AxisControl/GetTablePosition

Get the current position of a table.

Arguments:

- **table_name <string>** - The table to get position for.

Response:

- **position <** Coordinate **>** - The current position of the table.

```
{
    "module": "AxisControl",
    "cmd": {
        "func": "GetTablePosition",
        "args": {
            "table_name": "LeftTable"
```

(continues on next page)

(continued from previous page)

```
        }
    }
}

{
    "ret": {
        "position": [32.54, 102.034]
    },
    "status": "ok"
}
```

9.5.10 SetTriggerWindow

AxisControl/SetTriggerWindow

Configure a trigger window that will generate trigger pulses from the motion controller when moving through the window in y-direction. The length of the window is determined by the pulse distance and number of pulses.

For example, if the start position is 20 mm, the pulse distance is 10 um and the number of pulses is 10000, the window will be from 20 mm to 120 mm. The triggers will be generated irrespective of the direction of the motion.

Arguments:

- **table_name <string>** - The name of the table to set up trigger window for.
- **enable_triggers <bool>** - If TRUE triggers will be generated within the defined window, if FALSE no triggers will be generated. NOTE: If enable_triggers is FALSE, the other arguments are not needed.
- **start_pos <** Length **>** - *OPTIONAL*. The y-start position of the trigger window. This parameter is not needed if enable_triggers is FALSE.
- **pulse_dist <** Length **>** - *OPTIONAL*. The distance between each trigger pulse. This parameter is not needed if enable_triggers is FALSE.
- **num_pulses <number>** - *OPTIONAL*. The number of pulses to be generated. This parameter is not needed if enable_triggers is FALSE.

```
{
    "module": "AxisControl",
     "cmd": {
         "func": "SetTriggerWindow",
         "args": {
             "table_name": "table1",
             "enable_triggers": true,
             "start_pos": [20, "mm"],
             "pulse_dist": [10, "um"],
             "num_pulses": 10000
         }
     }
 }

 {
     "status": "ok"
 }
```

9.6 Event

9.6.1 AddListener

Event/AddListener

Add event(s) to receive on the calling socket via an event listener.

Arguments:

One of the below arguments must be given.

- **events <array of strings>** - *OPTIONAL.* Select which event categories to subscribe events from. Valid event categories are:
  - PHOTOHEAD_CONNECTIVITY
  - LIGHT_SOURCE_ISSUE
- **all <boolean>** - *OPTIONAL.* If set to true, all events will be sent. This will override the events list if given.

Valid event types are:

For PHOTOHEAD_CONNECTIVITY:

- CONNECTION_DOWN
- CONNECTION_UP
- LIMIT_EXCEEDED

For LIGHT_SOURCE_ISSUE:

- OVER_CURRENT_PROTECTION
- OVER_POWER_PROTECTION
- DIODE_TEMP_PROTECTION
- DRIVER_TEMP_PROTECTION
- DOOR_OPEN_PROTECTION
- FORMATTER_BOARD_TEMP_TOO_HIGH

```
{
   "module": "Event",
   "cmd": {
     "func": "AddListener",
     "args": {
       "events": ["PHOTOHEAD_CONNECTIVITY"]
     }
   }
}

{
  "status": "ok"
}
```

Events will be delivered to the listener like this:

- **category <string>** - The event's category.
- **type <string>** - The event type.

- **what <object>**
  - **message <string>** - The message describing the event.
  - **ip <string>** - The IP address of the device that triggered the event.

```
{
  "event": {
    "category": "LIGHT_SOURCE_ISSUE",
    "type": "DIODE_TEMP_PROTECTION",
    "what": {
      "message": "Light Source Diode temperature protection triggered on 192.168.0.
↪10",
      "ip": "192.168.0.10"
    }
  }
}
```

9.6.2 RemoveListener

Event/RemoveListener

Remove event listener on the calling socket.

Function removes all listeners.

```
{"module": "Event", "cmd":
    {"func": "RemoveListener"}
}

{"status": "ok"}
```

9.7 Measurement

9.7.1 ScanForPowerDevices

Measurement/ScanForPowerDevices

Scan for measurement devices that are compatible and return a list of these. The devices are identified by a unique ID. The ID must be used in the other measurement functions to identify the device to use (device argument).

Arguments:

- **devices <vector<string>>** - Devices found during scanning

9.7.2 MeasurePowerCurve

`Measurement/MeasurePowerCurve`

Measure power curve for a given photohead and light source driver using a connected measurement device.

Arguments:

- **device <string>** - Already connected device to use for measurement
- **ph_no <number>** - Photohead to interact with
- **driver_no <number>** - Light Source Driver to interact with
- **start_amplitude <number>** - Start amplitude
- **end_amplitude <number>** - End amplitude
- **step_amplitude <number>** - Step amplitude
- **measurement_area_mm2 <number>** - (Optional) Measurement area in mm2
- **total_area_mm2 <number>** - (Optional) Total area in mm2

Work Status:

Response:

- **max_power <number>** - Max power
- **curve_points <array>** - Array of curve points
  - **amplitude <number>** - Amplitude
  - **power <number>** - Power
  - **ocp <bool>** - Over Current Protection
  - **opp <bool>** - Over Power Protection
  - **otp_led <bool>** - Over Temperature Protection LED
  - **otp_driver <bool>** - Over Temperature Protection Driver
  - **ma_current <number>** - Current in mA
  - **percent_increase <number>** - Percent increase since previous point

9.7.3 PowerCalibration

`Measurement/PowerCalibration`

Create a calibration table with up to 40 points to calibrate amplitude to a given power for a given photohead and light source driver using a connected measurement device.

The table calculated will need to be loaded via command LightSource/LoadCalibrationTable.

Arguments:

- **device <string>** - Already connected device to use for measurement
- **ph_no <number>** - Photohead to interact with
- **driver_no <number>** - Light Source Driver to interact with
- **step <number>** - Milliwatt step size per 100 amplitude
- **max <number>** - Maximum power in milliwatts to end calibration at

- **accuracy <number>** - Accuracy in percent, when to accept a calibration point, if set too tight calibration will not be able to be completed. Must stop the function with stop action if so.
- **measurement_area_mm2 <number>** - *(Optional)* Measurement area in mm2
- **total_area_mm2 <number>** - *(Optional)* Total area in mm2

Work Status:

Response:

- **calibration_points <array>** - Array of 16 bit calibration factors that can be used for LightSource/LoadCalibrationTable
- **calibration_details <object>**
  - **amplitude <number>** - Uncalibrated amplitude
  - **optical_power <number>** - Power calibrated to
  - **calibration_point <number>** - Calibration factor

CHAPTER

TEN

API TYPES DESCRIPTION

10.1 System

10.1.1 ProductID

`System/ProductID`

String identifier defining product type.

Types of photoheads supported by this application:

- **"lb4k6"** = Luxbeam 4k6 based
- **"lb4k8"** = Luxbeam 4k8 based
- **"lb9k5"** = Luxbeam 9k5 based
- **"LRS_MCX_V1"** = LRS MCX V1
- **"LRS_MCX_V2"** = LRS MCX V2
- **"LRS_MCX_WX_NIR"** = LRS MCX WX NIR
- **"LRS_MCX_4K"** = LRS MCX 4k
- **"LRS_MCX_8KA"** = LRS MCX 8kA
- **"LRS_COMPACT"** = LRS Compact
- **"LRS_WQ_PLUS"** = LRS WQ Plus
- **"LLS30"** = LLS 30
- **"LLS30_BEAM"** = LLS 30 Beam
- **"LLS15"** = LLS 15
- **"LLS06"** = LLS 06
- **"LLS04"** = LLS 04
- **"LLS2500"** = LLS 2500

Emulated photoheads that can be used for testing:

- **"LRS_MCX_V2_EMU"** = Emulated LRS MCX V2
- **"LRS_MCX_4K_EMU"** = Emulated LRS MCX 4k
- **"LRS_MCX_8KA_EMU"** = Emulated LRS MCX 8kA
- **"LRS_COMPACT_EMU"** = Emulated LRS Compact

- **“LLS30_EMU”** = Emulated LLS 30
- **“LLS30_BEAM_EMU”** = Emulated LLS 30 Beam
- **“LLS15_EMU”** = Emulated LLS 15
- **“LLS06_EMU”** = Emulated LLS 06
- **“LLS04_EMU”** = Emulated LLS 04
- **“LLS2500_EMU”** = Emulated LLS 2500

Arguments:

- **<string>** - String identifying product type.

CHAPTER

ELEVEN

EXCEPTION DESCRIPTION

All error codes (error categories) are described here:

exception PROCESS_ERROR
: code 1 - Error in creating a sub-process (eg. luxapp). Check RAM and disk space and working directory integrity.

exception INVALID_ARGUMENT
: code 2 - Argument is not valid

exception NOT_SUPPORTED
: code 3 - Operation not supported

exception DO_NOT_EXIST
: code 4 - Object does not exist

exception ALREADY_EXIST
: code 5 - Object already exist

exception NOT_INITIALIZED
: code 6 - Object not initialized

exception ALREADY_RUNNING
: code 7 - Object already running

exception TIME_OUT
: code 8 - Operation timed out

exception ABORTED
: code 9 - Operation aborted

exception FILE_ERROR
: code 10 - File error

exception BITMAP_ERROR
: code 11 - Bitmap error

exception OUT_OF_MEMORY
: code 12 - Out of memory

exception NOT_ALLOWED
: code 13 - Operation not allowed in current state

exception PHOTOHEAD_COMMS_ERROR
: code 14 - Photohead communication error

exception AXIS_CTRL_ERROR
code 15 - Axis controller error

exception AXIS_CTRL_COMMS_ERROR
code 16 - Axis controller communication error

exception REGISTRATION_ERROR
code 17 - Registration error

exception FIRMWARE_UPGRADE_NEEDED
code 18 - Firmware upgrade needed

exception PREPRO_ERROR
code 19 - Preprocessing error

exception CONFIGURATION_ERROR
code 20 - Some settings are not compatible with each other, or are missing

exception HW_MISSING
code 21 - Hardware missing

exception VISION_ERROR
code 22 - Vision system error.

exception HW_NOT_SUPPORTED
code 23 - Hardware not supported

exception INTERNAL_ERROR
code 10000 - Unexpected error in the code, that should ideally never happen

11.1 JSON

```
{
"PROCESS_ERROR": {
    "number": 1,
    "comment": "Error in creating a sub-process (eg. luxapp). Check RAM and disk space␣
↪and working directory integrity."
},
"INVALID_ARGUMENT": {
    "number": 2,
    "comment": "Argument is not valid"
},
"NOT_SUPPORTED": {
    "number": 3,
    "comment": "Operation not supported"
},
"DO_NOT_EXIST": {
    "number": 4,
    "comment": "Object does not exist"
},
"ALREADY_EXIST": {
    "number": 5,
    "comment": "Object already exist"
```

(continues on next page)

(continued from previous page)

```
},
"NOT_INITIALIZED": {
    "number": 6,
    "comment": "Object not initialized"
},
"ALREADY_RUNNING": {
    "number": 7,
    "comment": "Object already running"
},
"TIME_OUT": {
    "number": 8,
    "comment": "Operation timed out"
},
"ABORTED": {
    "number": 9,
    "comment": "Operation aborted"
},
"FILE_ERROR": {
    "number": 10,
    "comment": "File error"
},
"BITMAP_ERROR": {
    "number": 11,
    "comment": "Bitmap error"
},
"OUT_OF_MEMORY": {
    "number": 12,
    "comment": "Out of memory"
},
"NOT_ALLOWED": {
    "number": 13,
    "comment": "Operation not allowed in current state"
},
"PHOTOHEAD_COMMS_ERROR": {
    "number": 14,
    "comment": "Photohead communication error"
},
"AXIS_CTRL_ERROR": {
    "number": 15,
    "comment": "Axis controller error"
},
"AXIS_CTRL_COMMS_ERROR": {
    "number": 16,
    "comment": "Axis controller communication error"
},
"REGISTRATION_ERROR": {
    "number": 17,
    "comment": "Registration error"
},
"FIRMWARE_UPGRADE_NEEDED": {
    "number": 18,
    "comment": "Firmware upgrade needed"
```

(continues on next page)

(continued from previous page)

```
},
"PREPRO_ERROR": {
    "number": 19,
    "comment": "Preprocessing error"
},
"CONFIGURATION_ERROR": {
    "number": 20,
    "comment": "Some settings are not compatible with each other, or are missing"
},
"HW_MISSING": {
    "number": 21,
    "comment": "Hardware missing"
},
"VISION_ERROR": {
    "number": 22,
    "comment": "Vision system error."
},
"HW_NOT_SUPPORTED": {
    "number": 23,
    "comment": "Hardware not supported"
},
"INTERNAL_ERROR": {
    "number": 10000,
    "comment": "Unexpected error in the code, that should ideally never happen"
}
```

}