**Client/Server interfaces for LasGo, DCx camera, and OceanOptics spectrometer**

**(and others later)**

I redefined an interface that should give us more flexibility and hopefully will be more compatible with Python or whatever else we plan to use.

Communication is over a persistent socket opened on monitored ports. Expect each “tool” to have its own port even if they are physically on the same computer.

**DCx Camera:** port 985

Once opened, communication is always a request/response sequence. The request and response use a common structure containing the action and minimal information as necessary.

#pragma pack(4)

typedef struct \_DCX\_MSG {

uint32\_t msg;

uint32\_t msgid;

uint32\_t rc;

uint32\_t data\_len;

} DCX\_MSG;

#pragma pack()

All parameters are 32 bit integers and packed in network order (ntohl and htonl).

For **requests**, the parameters are:

* **msg**: command or action to be executed. For the DCx camera, allowed values are:

#define DCX\_QUERY\_VERSION (1) /\* Return version of the server code \*/

#define DCX\_GET\_CAMERA\_INFO (2) /\* Return structure with camera data \*/

#define DCX\_ACQUIRE\_IMAGE (3) /\* Acquire image only (local storage) \*/

#define DCX\_GET\_IMAGE\_INFO (4) /\* Return info on the image \*/

#define DCX\_GET\_IMAGE\_DATA (5) /\* Transfer the actual image data \*/

* **msgid**: identifier associated with the command. Will be used if we implement queuing, but currently ignored.
* **rc**: generally, not used but may be used to pass a single parameter. Should be zero normally.
* **data\_len**: length (in bytes) of associated data to be sent, or zero if unneeded. If there is additional data, this informs the server of the number of bytes that should be expected in the subsequent (immediate) send(). Currently no commands transmit additional data.

The **response** to a request will have the same structure, with parameters

* **msg**: if no error, will be same as the request msg. If it does not match, consider as a fatal error.
* **msgid**: returned identifier associated with the command (for queued requests eventually)
* **rc**: return code from command. 0 if successful except for DCX\_QUERY\_VERSION (then version #)
* **data\_len**: length (in bytes) of associated data that will immediately follow. Client should allocate a buffer and recv() the data sent. Interpretation of the data depends on the command.

The request/response are in network order, but the no translation is used by default for the additional data sent either as part of the request or as part of the response. Managing interpretation of the byte order is left to the program.

**DCx CAMERA ACTIONS:**

**DCX\_QUERY\_VERSION:** No parameters. Returns version number embedded in the server code (1001 currently). It is important to verify the version number corresponds as this API may change rapidly.

**DCX\_GET\_CAMERA\_INFO:** No parameters. Returns a structure (in the second send()) containing details about the camera. The structure is defined as

#pragma pack(4)

typedef struct \_DCX\_STATUS {

char manufacturer[32]; /\* Camera manufacturer \*/

char model[32]; /\* Camera model \*/

char serial[32]; /\* Camera serial number \*/

char version[32]; /\* Camera version \*/

char date[32]; /\* Firmware date \*/

uint32\_t CameraID; /\* Camera ID (as set in EEPROM) \*/

DCX\_IMAGE\_TYPE color\_mode; /\* Monochrome or color mode \*/

uint32\_t pixel\_pitch; /\* Pixel pitch in um \*/

double fps; /\* Frame rate (frames per second) \*/

double exposure; /\* Current exposure (ms) \*/

double gamma; /\* Gamma value \*/

uint32\_t master\_gain; /\* Gains in non-linear range [0,100] \*/

uint32\_t red\_gain, green\_gain, blue\_gain;

uint32\_t color\_correction; /\* 0,1,2,4,8 ==> disable, enable, BG40, HQ, IR Auto \*/

double color\_correction\_factor;

} DCX\_STATUS;

#pragma pack()

**DCX\_ACQUIRE\_IMAGE:** No parameters. Acquire an image using the current acquisition parameters. The image is stored on the server side for subsequent readout, along with details of the acquisition.

**DCX\_GET\_IMAGE\_INFO:** No parameters. Returns a structure describing the acquisition parameters. The data is persistent on the server until a subsequent DCX\_ACQUIRE\_IMAGE command. Note that the memory\_pitch element is critical; this is the byte spacing between subsequent rows of the image, with potential for padding at the end of each line for memory alignment.

#pragma pack(4)

typedef struct \_DCX\_IMAGE\_INFO {

uint32\_t width, height; /\* Image width and height \*/

uint32\_t memory\_pitch; /\* Bytes between each row (allocate pitch\*height) \*/

double exposure; /\* Current exposure (ms) \*/

double gamma; /\* Gamma value \*/

uint32\_t master\_gain; /\* Gains in non-linear range [0,100] \*/

uint32\_t red\_gain, green\_gain, blue\_gain;

uint32\_t color\_correction; /\* 0,1,2,4,8 ==> disable, enable, BG40, HQ, IR Auto \*/

double color\_correction\_factor;

uint32\_t red\_saturate, green\_saturate, blue\_saturate; /\* Number saturated pixels \*/

} DCX\_IMAGE\_INFO;

#pragma pack()

**DCX\_GET\_IMAGE\_DATA:** No parameters. Returns a very large second send() containing the entire image (typically 3.93 MB for color images and 1.31 MB for monochrome). For color images, the sequence is BGR in memory.

I implemented a “test” server that does not query a physical camera but responds to all of these messages. It compiles and runs under Windows, and might work okay under Linux. There is a corresponding client that can access this test server, or access the actual camera in the lab (change the IP address of the server).

**Client-Server commands for LasGo (stage and laser control)**

These include a text based message interface as there generally will not be large data blocks to be transmitted (at least that was the original thought but not totally valid now). There is also a message based interface, but scripting languages will generally find the text based interface easier to implement.

Communication is over a persistent socket opened on monitored ports. Expect each “tool” to have its own port even if they are physically on the same computer.

**Structure Based Interface:** port 995

**Text Based Interface:** port 994

**Text message Interface:**

Commands are transmitted as simple text strings with arguments encoded as simple text. All commands are case insensitive and include both verbose versions and 2-character abbreviations. Code does not strictly check for argument validity on all commands and invalidly formatted / missing arguments will commonly be interpreted as zero.

All commands return a message on the same port when complete. Message formatted as a 6-digit message length (in total bytes), a colon, and the return message text. For example, the return from a QUERY\_POSITION command might be

000021:15.317 -28.002

The length is provided as information only to allow clients to ensure they have received all of the return message before interpreting the data (see QT command). Note that the inclusion of the length can be turned off via a flag in the code.

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Arguments** | **Return message** | **Notes** |
| QV  QUERY\_VERSION | [none] | integer version number | Version number continually incremented. Should verify that version is as expected before expecting commands to still be valid. |
| MV <xpos> <ypos>  MOVE\_QUEUED <xpos> <ypos> | positions in mm (double) | 0 on success, otherwise error code from Stage() call | Queues stage movement to specified position. Returns immediately. Stage precision is 0.001 mm. |
| BM <xpos> <ypos>  MOVE\_IMMEDIATE <xpos> <ypos> | positions in mm (double) | 0 on success, otherwise error code from Stage() call | Moves stage to specified position. Returns when complete. Stage precision is 0.001 mm. |
| GP  QUERY\_POSITION | [none] | x,y position (double) | Positions always relative to current origin |
| QO  QUERY\_ORIGIN | [none] | index “text” | Return index is an integer [0,7] for current coordinate system. The text is the associated text with that origin (from setup). Return message includes the double quotes around text. |
| SO [ index | “text”]  SET\_ORIGIN [ index | “text”] | index of desired origin[0,7] –**or-** text associated with the origin (from setup) | 0 on success, otherwise  -1 - index out of range  -2 - text not found | Sets the origin to be used for stage movement. Origins typically correspond to laser or characterization positions. If text given, comparison is case insensitive but otherwise exact. Quotes are required with text. |
| WA  WAIT | [none] | 0 on success, otherwise flags from AxisError() | Waits for all stage movement to complete before returning |
| AB [<timeout\_ms>]  ABORT [<timeout\_ms>] | optional timeout for stage access semaphore (default 100 ms) | 0 on success or return code from U500 abort command | Aborts any current move command. Due to multithread semaphore protection, normally waits to get access to stage. But will send abort after specified timeout even if control is not obtained. |
| PA [<timeout\_ms>]  PANIC [<timeout\_ms>] | optional timeout for stage access semaphore (default 100 ms) | 0 on success or return code from U500 abort command | Panic abort. Halts all stage motion, disables PSO, and disengages all motors (frees stage). See ABORT command for timeout behavior. |
| ST  STATUS | [none] | 14 parameters as text  rc with bit-flags  full status flag  x,y status flags  x,y relative position  x,y absolute position  x,y feedback position  x,y position error  x,y velocity  status flags are int, others are doubles | Not typically useful, but details are available if necessary |
| FS <xstart> <ystart> <xend> <yend> <velocity> <mm\_trig\_spacing> <num\_pts>  FLY\_SCAN <xstart> <ystart> <xend> <yend> <velocity> <mm\_trig\_spacing> <num\_pts> | Conditions for CHESS x-ray scan. Start/end are mm, velocity is mm/s, and spacing is mm | 0 on success or return code from server scan code (CHESS specific) | Existing code for X-ray scans. Unlikely to be used in the future, but kept since functional. |
| QT  QUERY\_TRIGGERS | [none] | List of positions for x-ray triggers in format  (-18.4287,97.0832) (including parentheses) separated by a space. | Allows encoding of best estimate for position each x-ray scan would correspond to on the wafer into scan metadata |