PH16 camera protocol

Vlad Iordăchescu, Radu Corlan

Version 2.3 February 2, 2014

Contents

1 Introduction

This document is a description of the second version of the control protocol for the Phantom high-speed cameras that support an Ethernet connection. At the time of writing, the supported models are the V16 series the MIDI family and the P4k. The protocol is largely independent of the camera model.

We aim to describe the protocol as implemented in the cameras at the time of writing. Therefore, it has to be assumed that the protocol or it's implementation may change in the future. While efforts will be made to maintain compatiblity, it cannot be assumed that full compatibility of future versions is maintained.

Communication interfaces of high-speed cameras serve two main purposes: they provide a means to control the operation of the camera; and they provide a means of downloading image and auxiliary data.

Each communication interface of the camera (ethernet, serial) supports at least one control stream. All these streams are active simultaneously. As they all act on the same underlying camera hardware, their responses are interdependent. The state of this hardware will be called "unit status" in the rest of the document

No "session" or similar locking mechanism is implemented. It is assumed that the application software can handle asynchronous changes in the unit status.

The communication protocol itself is "stateless". The only interaction between the various streams is through unit status changes.

Data streams can be created using the control stream commands `startdata` or `attach`. They are used to transfer large amounts of binary data. Only one data stream can be created for each interface at a given time. The serial interface does not support data streams.

To determine if a camera uses the new v16 protocol, or the old one (v7), examine the response to the discovery packet, which is different, or the the `info.pver` variable, which has a value of 16 for the new protocol, and is not implemented in ph7.

2 Establishing Control Streams

2.1 Ethernet

On Ethernet, control streams are implemented as TCP stream sockets. The camera will accept TCP connections to port 7115. Each socket thus created constitutes a control stream.

The default port number (7115) may change. The current value for each camera, as well as the IP addresses of all cameras connected to a network can be determined using the discovery protocol (see Section 6 below).

Several simultaneous connections are accepted. Simultaneous control of the camera through several connections is possible; this assumes that the controlling applications are capable of handling all the unit state changes correctly. A more realistic setup would consist of one controlling application for each camera and other utilities spread on the network that are used to monitor the status of the cameras.

2.2 Serial Interface

The serial port has the simplest establishment procedure: a control stream is considered to be established between the camera and whichever device is connected to the port at all times. No handshaking is used on the serial port: the camera accepts commands at line speed, and the connecting device is supposed to do the same. Data format is 8N1, and the default connection speed is 38400 baud. The speed can be set using a protocol command, and this change will remain in effect until the camera is rebooted. No error checking is implemented.

3 Control Stream Syntax

All communication over the control streams is done in text (ASCII) format. The commands are formatted so that it is reasonable for a human to type commands and interpret the results.

The control stream command interpreter reads *command lines* and produces *response lines*. For each command line there is exactly one reponse line. Receiving a newline causes the command interpreter to attempt to execute all input since the last newline as a command. After the response (or error) is generated, the command interpreter returns to it's base state.

Command lines are limited to a total length of 65536 bytes (including the newline). All response lines are guaranteed not to exceed the same length.

3.1 Command and Response Lines

A command line is a character string that begins with the command name and ends with a *newline*. A `CR` character or the `CRLF` sequence are considered newlines.

Newlines can be escaped using the `'\'` character. The `\newline` sequence is considered white space when the command line is read.

Response lines generated by the camera always end in `CRLF`. If a response line is likely to exceed 80 charaters in length, it is split using `\CRLF` sequences for lisibility.

3.2 Data Types

Various data types are represented as follows.

white space Any sequence of spaces, tabs and `\newline` sequences constitues white space and is interpreted as a single space character;

command name A sequence of letters and digits starting with a letter. In general, all commands have lower-case names.

Example: `set`

unsigned integer A sequence of decimal digits. While most integer values are 32-bit numbers, some can be longer. The protocol does not enforce any size limit.

Example: `10000`

signed integer A sequence of decimal digits optionally starting with a minus sign.

Example: `-10000`

hex number A sequence of hexadecimal digits starting with `0x`.

Example: `0x1fffff`

floating point number A floating point number.

Examples: `-2.5, 1.2e3`;

character string A sequence of characters enclosed in double quotes. String variables representing IP addressed are 16 chars long. The `meta.comment` and `meta.xset` string variables are 4096 chars long. All other string variables are 256 chars long.

Example: `"camera name"`;

resolution A sequence of the form: `<xres>x<yres>`, where `<xres>` and `<yres>` are unsigned integers.

Example: `1280x800`;

variable name One or more sequences of letters and digits each starting with a letter, separated by `'.'`.

Examples: `info`, `info.name`;

flag list A sequence of the form: `{ <flagname> ... }`, where `<flagname>` is a three-character string. The valid flags are dependent on the specific comand or variable used.

Example: `{WTR ACT}`;

tagged list A sequence of the form:
`{ <variable name> : <value> | <tagged list> , ...}`.

Examples of tagged lists are:

```
{defc:{meta:{w:1}}}, info:{name:"rearview camera"}}
```

```
{{res:1024x768}, frcount: 500}
```

4 The Unit Structure

From the point of view of the camera interface programmer, the unit structure is the sole repository of the camera's status. All changes in the various operating parameters, as well as the actual acquisition status are trasferred to and from the user by means of changes of variables in this structure.

The values of these variables are read and altered by the following concurrent processes:

- A command interpreter for each control stream;
- The *machine process*, which transfers the changes of the variables to the camera hardware and the changes in hardware status to the unit structure.

All operations on the unit structure are atomic in the sense that if a given command changes more than one variable, no other command (e.g. from another control stream or the machine process) is executed until all the changes are made.

The unit structure is hierarchical, similar to a C struct. Operations can be performed on either leaf nodes (single variables) or whole branches of the struct. A detailed description on ways to acces the structure is in the `get` and `set` commands section (5.1 and 5.2).

4.1 Unit Structure Members

The unit structure has the following members (top-level substructures):

`info`	Various information about the camera configuration.
`hw`	Structure holding factory-adjusted calibration settings, not to be changed during normal camera use; It may become protected and/or invisible in future versions.
`meta`	Descriptive metadata that gets saved with each cine.
`cam`	Global settings for the camera, that aren't tied to a particular cine.
`auto`	Description of automatic actions performed by the camera at the end of a cine recording.
`eth`	Global settings concerning the ethernet interface.
`video`	Global settings concerning the video out and OSD.
`irig`	Status of the timbebase and range data acquisition system of the camera.
`mag`	Structure containing status information regarding the cinemag.
`cf`	Status information for any attached storage devices as Compact-FLASH cards, SSDs, HDDs, etc.
`usets`	Information regarding the user settings sets.
`defc`	A special cine structure used to provide default parameters to cines and change the current acquisition parameters of the camera.

`cine table`	Table of 'cines', i.e. structures holding the acquisition parameters and status of image recordings.

4.2 The Info Structure

The `info` structure contains "fixed" information about the camera configuration. Most of it's members are read-only.

4.2.1 Sensor Information

Name	Type	Access	Description
`info.sensor`	uint	r/o	Type of sensor used.
`info.snsversion`	uint	r/o	Version number of the sensor.
`info.cfa`	uint	r/o	Type of color filter array deposited on the sensor.
`info.filter`	uint	r/o	Additional filter used.

4.2.2 Versions and Identification

Name	Type	Access	Description
`info.hwver`	uint	r/o	Current hardware version.
`info.kernel`	uint	r/o	Current kernel version.
`info.swver`	uint	r/o	Current firmware version.
`info.xver`	uint	r/o	Current FPGA version.
`info.model`	string	r/o	Camera model.
`info.pver`	uint	r/o	Protocol version (16).
`info.sver`	uint	r/o	System (kernel+filesystem) release number.
`info.fver`	uint	r/o	Firmware release number.
`info.serial`	uint	r/o	Camera serial number.
`info.name`	string	r/w, save	Camera name.

4.2.3 Capabilities

Name	Type	Access	Description
`info.features`	string	r/o	A string of tokens separated by spaces which indicate if the camera supports certain features.
`info.imgformats`	string	r/o	Image formats supported by the `img` command.
`info.videosystems`	string	r/o	List of formats supported on the video output. The formats are specified by the codes used by the `video.system` variable.
`info.maxcines`	uint	r/o	Maximum number of cines that can be allocated.

`info.xmax`	uint	r/o	Maximum horizontal resolution (pixels).
`info.ymax`	uint	r/o	Maximum vertical resolution (pixels).
`info.xinc`	uint	r/o	Horizontal increment; valid horizontal resolutions are a multiple of `xinc`.
`info.yinc`	uint	r/o	Vertical increment; valid vertical resolutions are a multiple of `yinc`.
`info.winx`	uint	r/o	Windows can start at multiples of `winx` pixels from the left sensor edge.
`info.winy`	uint	r/o	Windows can start at multiples of `winy` pixels from the top sensor edge.
`info.kernsz`	uint	r/o	Size of acquisition kernel (in bytes); all image memory is allocated internally in kernel units. `kernsz` is used to transform that into bytes.
`info.memsz`	uint	r/o	Size of installed memory in megabytes.
`info.cinemem`	uint	r/o	Total amount of image memory available for cine storage in megabytes.
`info.mdepths`	hex	r/o	Bit mask describing the available memory depths in the camera. If the n-th bit of this field is set, then image data can be acquired using an n bits pixel size.
`info.expdead`	uint	r/o	Minimum interval between two exposures; the shutter has to stay "closed" for this time, so the maximum exposure time is `1/rate - expdead`.
`info.minexp`	uint	r/w, save	Minimum exposure the camera is capable of, in ns.
`info.xblock`	uint	r/o	Number of `x` pixels the camera reads in a clock cycle.
`info.yblock`	uint	r/o	Number of `y` pixels the camera reads in a clock cycle.
`info.pixps`	uint	r/o	Time to read one pixel in picoseconds.
`info.rotps`	uint	r/o	Row overhead time in picoseconds.
`info.fotps`	uint	r/o	Frame overhead time in picoseconds.
`info.minfrate`	uint	r/o	Minimum framerate.
`info.maxrate`	uint	r/o	Maximum framerate.
`info.tmodel`	uint	r/o	Timing model used by the camera. Valid values = 4.
`info.magtp`	uint	r/o	Maximum pixel troughoutput that the camera can send to the cinemag interface, in pixels/sec.
`info.rtobyteps`	uint	r/o	Duration of one byte in picoseconds.
`info.rtopacket`	uint	r/o	Size of a packet in bytes.
`info.rtopacketovhead`	uint	r/o	Overhead per packet in bytes.

`info.rtofrovhead`	uint	r/o	Overhead per frame in bytes.
`info.rto_channels`	uint	r/o	Number of RTO channels.

4.2.4 Feature string tokens

`bref`	The camera supports internal CSR (black reference) operations.
`atrig`	The camera has an image-based autotrigger system.
`blk4`	The nominal black level of raw corrected images is at 4/256 of full scale.
`v444`	The camera support 4:4:4 3G-SDI outputs.
`shtr`	The camera has a capping shutter installed.
`lowexp`	The camera has the FAST option installed, allowing minimum exposures of under 1μs.
`hqmode`	The camera supports the high-quality acquisition mode.
`ximg`	The camera supports the `ximg` command.
`edr`	Camera supports double-slope exposures (edr).
`burst`	The camera supports frame burst mode.
`genlock`	The camera has a genlock system.
`decimate`	The camera allows frame decimation.
`ramp`	The camera allows frame rate ramping during recording.
`aexp`	The camera has an autoexposure system.
`mag`	The camera has a cinemag socket.
`cf`	The camera has a file storage device (CF card, SSD, etc) port.
`earlyimg`	The camera supports reading numbered frames as soon as a cine is triggered.
`attach`	The camera supports the `attach` command and corresponding method of establishing data sockets.
`notify`	The camera supports the `notify` command that enables asynchronous state change messages.
`v4k`	The camera support 4k video output on dual 3G-SDI outputs.
`log`	The camera support log modes for the video outputs.
`dualp`	The camera supports dual video pipes.
`aux2`	The camera supports aux2mode.
`mm24`	The camera supports 24-axes multi-matrix color correction.
`wide`	The camera supports anamorphic desqueeze (`video.widescreen`).
`audio`	The camera supports audio i/o.

4.2.5 Color correction

Name	Type	Access	Description
`info.basechroma`	float	r/o	Default chroma gain used when the color matrix is off.

info.baseei	uint	r/o	Base Exposure Index, in ISO units. Exposing for this index should result in a properly exposed image with the camera at minimum video gain, a neutral tone curve and a gamma of 2.2.

4.2.6 Camera status monitoring

Name	Type	Access	Description
info.snstemp	int	r/o	Sensor die temperature.
info.tepower	int	r/o	Amount of power used by the sensor thermoelectric cooler as a percentage of full cooling power. Negative power levels indicate that the sensor is being heated instead of being cooled.
info.camtemp	int	r/o	Camera temperature measured inside the body.
info.fanpower	int	r/o	Fan speed as a percentage of maximum speed.
info.batti	int	r/o	Battery current in milliamps.
info.battv	uint	r/o	Battery voltage in millivolts.
info.battstate	uint	r/o	Battery status: 0=no battery, 1=charging, 2=full, 3 =discharging, 4=low.
info.batttimer	uint	r/o	Number of seconds until camera will turn down by itself, 0=stay on until battery empty.
info.genlockstat	uint	r/o	State of genlock system: 0=poweroff, 1=idle, 2=measure line length, 3=wait for line lock, 4=measure field length, 5=enter field lock, 6=force field lock, 7=field lock check, 8=locked, 9=bad signal, 10=locked and time code present.

4.3 The Preset Structure

The preset structure contains sets of user-alterable strings that can be used as presets for the color correction algorithm.

Name	Type	Access	Description
preset.tone1	string	r/w	Storage for tone curve presets.
preset.tone2	string	r/w	Storage for tone curve presets.
preset.tone3	string	r/w	Storage for tone curve presets.
preset.tone4	string	r/w	Storage for tone curve presets.
preset.matrix1	string	r/w	Storage for color matrix presets.
preset.matrix2	string	r/w	Storage for color matrix presets.

`preset.matrix3`	string	r/w	Storage for color matrix presets.
`preset.matrix4`	string	r/w	Storage for color matrix presets.
`preset.filter1`	string	r/w	Storage for color filter presets.
`preset.filter2`	string	r/w	Storage for color filter presets.
`preset.filter3`	string	r/w	Storage for color filter presets.
`preset.filter4`	string	r/w	Storage for color filter presets.

The related `preset` command copies these values in `video.adj`.

4.4 The Cam Structure

The cam structure contains global user-alterable settings that aren't tied to a particular cine.

4.4.1 Camera Sync Options

Name	Type	Access	Description
`cam.syncimg`	uint	r/w, save	Frame sync mode: when 0, the camera free-runs; when 1, it locks to the fsync input; when 2, it locks to the irig timebase; when 3, it locks to the video frame rate.
`cam.frdelay`	uint	r/w, save	Delay between the sync reference moment and the start of exposure, in ns.
`cam.rtoen`	uint	r/w, save	1= enable the real time output, 0= disable.
`cam.rtotfr`	uint	r/w, save	RTO test frame size in bytes.
`cam.membpp`	uint	r/o	Bit depth into image memory. Can be changed via the `acqmode` command.

4.4.2 Global Camera Options

Name	Type	Access	Description
`cam.trigpol`	uint	r/w, save	Trigger polarity; when 0, the camera triggers on the falling edge of the trigger signal; when 1, on the rising edge.
`cam.trigfilt`	uint	r/w, save	Minimum time in μs the trigger signal has to be asserted continuously in order to be recognised. In addition, the signal needs to have been continuously unasserted for at least four times this amount of time for a trigger to be accepted.

cam.startonacq	uint	r/w, save	When 1, the camera starts in waiting for trigger mode; when 0, it starts in preview mode to save power.
cam.aux1mode	uint	r/w, save	Function of the aux1 signal (pin 6): 0=strobe, 1=event, 2=memgate, 3=fsync.
cam.aux2mode	uint	r/w, save	Function of the aux2 signal (pin 5): 0=ready, 1=strobe, 2=aes/ebu out.
cam.tsformat	uint	r/w, save	Timestamp format used in future recordings, as described in Section 5.12 .
cam.tcmode	uint	r/w, save	Timecode mode(0=irig, 1=smpte).
cam.master	uint	r/w, save	Serial number of the sync master. not used by the camera.
cam.apoffdis	int	r/w, save	Auto poweroff disable.
cam.longready	uint	r/o	Ready becomes inactive at end of recording instead of trigger time.
cam.cines	uint	r/o	Number of recordable cines (partitions) as generated by the partition command.
cam.dark	int	r/w, save	Close (1) / Open (0) the mechanical shutter.
cam.tsetsns	uint	r/o	Sensor temperature set point.
cam.tsetcam	uint	r/o	Camera temperature set point.
cam.tz	uint	r/w, save	Difference in seconds between local time and UTC. Used when displaying local time on the OSD. All time values in the protocol are UTC.

4.5 Camera Autosave and Continuous Recording

At the end of a recording, if auto.flashsave is set, the camera tries to save the cine to a cinemag; if auto.filesave is set, the camera tries to save the cine to an internal disk; if auto.videoplay is set, the camera will play the cine on the video output; if auto.acqrestart is set then, if a flashsave or a filesave were also requested and were performed succesfully, or no such operation was requested, the cine is deleted and recording is resumed.

For those cameras equipped with a capping shutter, an automatic black reference mode is available. If the auto.bref field is set, a black reference operation is automatically performed before a cine starts recording.

4.5.1 Auto-ops in multi cine mode

When the camera is configured to operate in multi cine mode, once the recording of a cine ends, the camera queues the corresponding automatic operations and executes them in sequence. Thus it is possible to perform the automatic operations for the just recorded cine, while continuing to record using the next available cines. If the auto.acqrestart flag is set, then, when all automatic operations involving a cine are

completed, that cine is marked as reusable. The cine will be deleted and reused when no other recordable cine is empty.

For example, if the camera has `C1...C5` configured, the recording starts in `C1`. When `C1` is triggered, the automatic operations for `C1` are pushed in the `auto queue`, and the recording starts in `C2`. When `C2` is triggered, the automatic operations for `C2` are pushed in the `auto queue`, and the recording starts in `C3`, and so on. When `C5` is also triggered, if there was enough time to finish the automatic operations involving `C1`, and the `auto.acqrestart` flag was set, then `C1` is already available and the recording continues in `C1`.

Name	Type	Access	Description
`auto.videoplay`	uint	r/w, save	When non-zero, the camera will play back the recorded cine on video immediately after the recording ends and the autosave operations are finished. A `rec` command or the pretrigger signal will abort a video play operation.
`auto.flashsave`	uint	r/w, save	When non-zero, a camera will save the just-recorded cine to a cinemag. If the cinemag fills up (or is not installed), this flag is cleared.
`auto.filesave`	uint	r/w, save	When non-zero, the camera will save the just-recorded cine to a attached storage device. If an error occurs during save, this flag is cleared. When both `flashsave` and `filesave` are set, a file save operation is not performed if the flash save was successful.
`auto.acqrestart`	uint	r/w, save	When non-zero, the camera will restart acquisition after a sucessfull automatic save.
`auto.bref`	uint	r/w, save	When non-zero, the camera will take a new black reference (CSR) at the start of each recording.
`auto.firstframe`	int	r/w, save	The first frame of the region that is to be saved and/or played on video.
`auto.lastframe`	int	r/w, save	The last frame of the region that is to be saved and/or played on video.

auto.loops	uint	r/w	A non-zero value is the loop counter for video play back. When the value is zero, the behavior of the camera is determined by the value of the auto.acqrestart field as follows: if auto.acqrestart is zero, then the video playback loops forever; if auto.acqrestart is non-zero, then the video is played back only once and afterwards the acquisition is restarted. A non-zero value N means to play the video N times.
auto.speed	uint	r/w	Speed of video play back (number of times each frame will get played).
auto.progress	uint	r/o	The number of frames still to do by the auto process.
auto.bref_progress	uint	r/o	The bref command accumulates the black reference for a total of eight frames. This is number of frames left.
auto.trigger	structure	r/w	Image-based autotrigger data. Described in a separate document.
auto.trigger.x	int	r/w	X coordinate of the center of the autotrigger region, relative to the center of the image
auto.trigger.y	int	r/w	Y coordinate of the center of the autotrogger region, relative to the center of the image
auto.trigger.w	uint	r/w	Width of the autotrigger region. When the width is zero, a default value will be used. the default value is one quarter of the image width or 128 pixels, whichever is larger.
auto.trigger.h	uint	r/w	Height of the autotrigger region. When the height is zero, a default value will be used. the default value is one quarter of the image height or 16 pixels, whichever is larger.

auto.trigger.threshold	uint	r/w	Amount a pixel value must change in order to be counted as an active pixel for autotrigger purposes. A value of 100 would require a change of approximately half of the full swing of the camera. A typical threshold setting would be 10.
auto.trigger.area	uint	r/w	Percentage of the area of the autotrigger region that must be active in order for an autotrigger event to be generated. A typical percentage value is 10.
auto.trigger.speed	uint	r/w	Number of frames between updates of the autotrigger reference memory. A value larger than 1 allows the trigger to activate on slower events. Image changes are evaluated over a time interval of auto.trigger.speed/frame_rate.
auto.trigger.mode	uint	r/w	Operating mode of the autotrigger.

4.6 The defc Structure.

The most important variables that control the camera image capture process, such as frame rate and exposure, are kept in the defc structure, named after the "default cine" of ph7.

Though patterned after a cine structure, defc is fundamentally different: it contains the parameters used by the camera "now", and changes to it control the current and future recordings, while the values in a cine structure from the cine table show what the camera was set to in the past, when the respecive cine was recorded.

4.6.1 Defc Structure Variables

Name	Type	Access	Description
defc.res	resolution	r/w, save	Image size in pixels.
defc.rate	float	r/w, save	Frame rate in pictures per second.
defc.exp	uint	r/w, save	Exposure time, in ns.
defc.edrexp	uint	r/w, save	EDR exposure time in ns.
defc.ptframes	uint	r/w, save	Number of post-trigger frames.
defc.shoff	uint	r/w, save	Shutter off: Always use maximum possible exposure and minimum straddle time.
defc.ramp	string	r/w,save	Frame rate ramping specification string.
defc.bcount	uint	r/w, save	Number of frames per burst. When 0 or 1, the frame burst mode is disabled.

`defc.bperiod`	uint	r/w, save	Time interval in *ns* between successive frames in a burst.
`defc.hqenable`	uint	r/w, save	Enable HQ acquisition mode.
`defc.decimation`	uint	r/w	Ratio between the frame rate going into camera RAM and the rate being output on RTO or the cinemag interface.
`defc.frcount`	uint	r/o	Number of frames that can be stored in a cine (either the currently active cine, or the first cine in the table that is available for recording in case the preview cine is the currently active cine).
`defc.frsize`	uint	r/o	Amount of space required to store a frame of the size specified in `defc`, in kernels. Dividing the value of a cine's `frspace` value to `defc.frsize` yields the capacity of that cine in frames.
`defc.aexpmode`	uint	r/w, save	Autoexposure mode: 0=off, 1=average, 2=spot, 3=center-weighted.
`defc.aexpcomp`	float	r/w, save	Autoexposure compensation in stops. Positive values increase the exposure.
`defc.meta.ox`	int	r/w, save	X offset of window from the camera frame center for application software crop and scale, in camera pixels.
`defc.meta.oy`	int	r/w, save	Y offset of window.
`defc.meta.w`	int	r/w, save	Window width in camera pixels.
`defc.meta.h`	int	r/w, save	Window height in camera pixels.
`defc.meta.ow`	int	r/w, save	Window width after software crop and scale in pixels.
`defc.meta.oh`	int	r/w, save	Window height after software crop and scale in pixels.
`defc.meta.crop`	int	r/w, save	Enable software crop.
`defc.meta.oh`	int	r/w, save	Enable software scale.

4.7 Cine Table

The cine table is an array of *cine structures*, each associated to a memory region used to store images and ancillary data such as time stamps. Cine `c0`, the preview cine, is a special case in that the short memory region associated with it is only used for reading live images. Cines `c1`, `c2` etc. are associated with normal RAM partitions, while cines `fc0`, `fc1`, etc. are associated with cinemag takes.

The number of RAM cine structures in the cine table is `info.maxcines`, of which one is the preview cine `c0`, and the next `cam.cines` ones have a memory partition associated with them and can be used to record and play images. The rest of the cines in the table are *invalid*, and do not participate in the camera operation.

The memory partitioning, and implicitly the number of valid cines, can be changed with the `partition` command. Cinemag cines are not allocated explicitly, but are automatically created when recording cinemag takes.

Most variables in the cine structures of the cine table hold copies, taken at the time when the cine was recorded, of the variables of the same name from the structures that control the camera, such as `defc`, `cam`, `video` or describe it (`info`, `meta`). Other few variables describe the memory partition itself, and it's state in the recording process.

When a memory partition does not contain image data, i.e. the cine state is `RDY` or `INV`, the content of the associated cine structure (except the state and allocation variables) is meaningless: the values are left over from either the camera initialisation, or a previous recording.

As a cine becomes active, i.e. it starts recording frames, and whenever the defc structure changes while it is active it's variables are updated from the defc structure, thus reflecting the camera setup at the time of recording. As other values in the cine structure become known (such as the trigger time, or the current number of frames in the cine), they are updated.

At the end of the recording, when the cine becomes "stored", all values in the structure are filled in. In the stored state, the only variables that can be changed are the `in` and `out` points for playback, and the image processing parameters from `c#.adj`.

Finally, when the cine is eventually deleted, the variables keep their values, but these values become meaningless again, as they refer to the recording which has just been deleted.

4.7.1 Status Variables

Name	Type	Access	Description
`c#.state`	flag list	r/o	Cine state.
`c#.frcount`	uint	r/o	Number of frames recorded.
`c#.firstfr`	int	r/o	Index of the first frame in the cine that can be played.
`c#.lastfr`	int	r/o	Index of the last frame in the cine that can be played.
`c#.format`	int	r/o	For non-RAM cines, the format the cine was saved in, which is also the only format it can be downloaded in. For RAM cines the values of this field is undefined, and they cab be downloaded in any format supported by the camera.

`c#.in`	int	r/o	Current in point for cine playback. The values of the in and out points are set to the first and last image in the cine when the cine becomes stored. They can later be changed to reflect the interesting region of the cine.
`c#.out`	int	r/o	Current out point for cine playback.

4.8 Cine States

The variable `c#.state` (of *flag list* type) holds the status flags for the cine. When a cine is invalid (unallocated), `state` takes the value `{"INV"}` and no other flags are ever present.

`INV` The cine is invalid; it has no memory allocated, nor does it participate in any way in camera operations.

`STR` The cine contains a complete, valid recording.

`WTR` The camera is currently recording this cine, and waiting for trigger.

`TRG` A trigger has been received and accepted for this cine.

`RDY` The cine is ready to receive a recording; `RDY` and `STR` cannot be present at the same time.

`DEF` If this flag is set, when acquisition starts into this cine, the acquisition parameters are first copied from the default cine, `defc`. In ph16, all cines have the `DEF` flag set at all times.

`ABL` If this flag is set, the cine can accept a trigger.

`PRE` This flag marks a special cine that is used to obtain live preview images when all the other cines are full. Normally, `c0` and only `c0` has the `PRE` flag set. The preview cine never has any of `ABL`, `WTR`, `TRG` or `STR` set.

`ACT` This flag marks the active cine, e.g. the cine into which images are acquired; only one cine can be active at any time.

`REU` The cine content has been saved, so it can be deleted by the camera if the auto.acqrestart option is set.

4.8.1 Memory Allocation

Name	Type	Access	Description
`c#.start`	hex	r/o	Start address of the image buffer (in kernels).
`c#.len`	hex	r/o	Length of the cine buffer (in kernels). The start and len are mostly used for debugging purposes.

`c#.frsize`	uint	r/o	frame size at the cine's resolution, in kernels. This value is invalid if the cine is not active or stored.
`c#.frspace`	uint	r/o	Cine memory region capacity in kernels. Dividing this value by `defc.frsize` yields the cine capacity in frames.

4.8.2 Trigger Time

Name	Type	Access	Description
`c#.trigtime.secs`	uint	r/o	Time when cine was triggered, in seconds since 1970.
`c#.trigtime.frac`	uint	r/o	Fraction of second of the trigger time (μs).

4.8.3 Acquisition Parameters

The acquisition parameter fields that match the ones in `defc` reflect the state of the latter during the recording. These fields are read-only.

4.8.4 Cam Substructure

The `c#.cam` substructure contains a read-only copy of the camera settings in the `cam` structure (see section 4.4). This copy is done at the time the cine recording ends, and reflects the parameters that were used during recording.

4.8.5 Info Substructure

The `c#.info` substructure contains a read-only partial copy of the `info` structure (see section 4.2). This copy is done at the time the cine recording ends. The purpose of this structure is to be able to identify the originating camera of a cine residing on a cinemag.

4.8.6 Adj Substructure

The `c#.adj` substructure contains a of the `video.adj` structure (see section 4.12). This copy is done at the time the cine recording ends, and reflects the video output adjustement in place at the end of recording. The substructure members can be modified afterwards and the changes will reflect on the video output when the cine is played. For a cinemag cine, the values can be changed but such changes are lost at poweroff, as the changes are not written back to the cinemag.

4.8.7 Meta Substructure

The `c#.meta` substructure contains a superset of the `meta` structure (see section 4.8.7). The extension fields are listed below.

Name	Type	Access	Description
`c#.meta.pbrate`	float	r/o	Default playback rate of cine, determined by the video format at the time of the recording.
`c#.meta.tcrate`	float	r/o	Time code rate, determined by video format.
`c#.meta.uuid`	string	r/o	Unique identifier for the cine.
`c#.meta.system`	uint	r/o	Value of the video.system field at the time of recording.
`c#.meta.trigtc`	string	r/o	Tigger timecode in hh:mm:ss.ss format.
`c#.meta.pax`	uint	r/o	Production area rectangle width. This and the next 9 variables are copies of the corresponding fields of the `video` structure, made at the time of recording.
`c#.meta.pay`	uint	r/o	Production area rectangle height.
`c#.meta.paox`	uint	r/o	Production area X offset from center.
`c#.meta.paoy`	uint	r/o	Production area Y offset from center.
`c#.meta.vox`	uint	r/o	Video output scaling (X offset from center).
`c#.meta.voy`	uint	r/o	Video output scaling (Y offset from center).
`c#.meta.vow`	uint	r/o	Video output scaling (output width).
`c#.meta.voh`	uint	r/o	Video output scaling (output height).
`c#.meta.vw`	uint	r/o	Video output scaling (input width).
`c#.meta.vh`	uint	r/o	Video output scaling (input height).
`c#.meta.ox`	uint	r/o	Software image scale and crop (X offset from center). This and the next seven variables come from `defc.meta`.
`c#.meta.oy`	uint	r/o	Software image scale and crop (Y offset from center).
`c#.meta.ow`	uint	r/o	Software image scale and crop (output width).
`c#.meta.oh`	uint	r/o	Software image scale and crop (output height).
`c#.meta.w`	uint	r/o	Software image scale and crop (input width).
`c#.meta.h`	uint	r/o	Software image scale and crop (input height).
`c#.meta.crop`	uint	r/o	Enable image crop by software.
`c#.meta.resize`	uint	r/o	Enable image scale by software.
`c#.meta.resize`	uint	r/o	Enable image scale by software.
`c#.meta.gps`	string	r/o	Copy of irig.gps made at the time the cine recording ended.

This copy is done at the time the cine recording ends, and reflects the meta values in place at the time of recording.

4.9 Irig Structure

The read-only irig structure returns the status of the camera timebase and range data acquisition unit. It is updated about once every second.

Name	Type	Access	Description
`irig.sec`	uint	r/o	Current time of the camera timebase in seconds since 1970.
`irig.yearbegin`	uint	r/o	Number of seconds since 1970 to the beginning of the current year; used to add year information to the irig time stamps.
`irig.flags`	flag list	r/o	Status flags for the timebase unit; see below.
`irig.signal`	string	r/o	Textual description of the incoming time code signal.
`irig.gps`	string	r/o	Current position and status info as received from a GPS receiver. It contains the GPRMC, GPGGA and GPGSA messages sent by the receiver.
`irig.range`	string	r/o	String representation of the current range data being received.

4.9.1 Irig Flags

`NUL` no other flag is set.
`LCK` timebase is locked to the irig source.
`MOD` irig source is modulated.
`EVT` event signal is active.
`RNG` range data is active.

4.10 Meta Structure

The meta structure contains descriptive metadata that is saved along with a cine.

Name	Type	Access	Description
meta.name	string	r/w	A 256 bytes string used to name the recorded cines. Succesive names are automatically generated by appending an index to meta.name. If meta.name already contains a trailing numeric value, this value is used as a start value for the index.
meta.lens	string	r/o	focal length range and aperture range of a supported "intelligent" lens. When no active lens mount is fitted, or no lens is mounted, this field is set to a null string.
meta.fstop	float	r/o	current fstop value of the fitted lens.
meta.flen	float	r/o	current focal length value of the fitted lens.
meta.comment	string	r/w	a 4 Kbytes comment string saved with the cine.
meta.xset	string	r/w	a 4 Kbytes string used for storing external applications settings.

4.11 Hw Structure

The harware setting structure contains factory-adjusted calibration settings, not to be changed during normal camera use; It may become protected and/or invisible in future versions. In general the variable meanings and values are specific to particular camera models. Some of the variables may be ignored by some camera models.

4.11.1 Bias voltages

Name	Type	Access	Description
hw.vpix	uint	r/w, save	Millivolts
hw.vpixl	uint	r/w, save	
hw.vmemh	uint	r/w, save	
hw.vmeml	uint	r/w, save	
hw.vresh	uint	r/w, save	
hw.vresl	uint	r/w, save	
hw.vresds	uint	r/w, save	
hw.dcblack	uint	r/w, save	
hw.bbias	uint	r/w, save	
hw.tbias	uint	r/w, save	
hw.trbias	uint	r/w, save	
hw.brbias	uint	r/w, save	

4.11.2 Sensor Timing Settings and Options

Name	Type	Access	Description
hw.vmlpred	uint	r/w, save	
hw.prepw	uint	r/w, save	
hw.presmpd	uint	r/w, save	
hw.smppw	uint	r/w, save	
hw.smpvmhd	uint	r/w, save	
hw.vmhresd	uint	r/w, save	
hw.respw	uint	r/w, save	
hw.reswait	uint	r/, savew	
hw.expadj	uint	r/w, save	
hw.sel2w	uint	r/w, save	
hw.shkolmode	uint	r/w, save	
hw.reslong	uint	r/w, save	
hw.snsflags	uint	r/w, save	
hw.nsfload	int	r/w, save	
hw.colload	int	r/w, save	
hw.outload	int	r/w, save	

4.11.3 Camera Timing Settings and Options

Name	Type	Access	Description
hw.freqtrim	int	r/w, save	Adjust the frequency of the internal time base, in units of about 1 part per billion. The total range is around +/- 10ppm.
hw.irigphase	int	r/w, save	Adjust the phase between unmodulated irig in and irig out, in ns; range is around +/- 100μs.
hw.irigmodphase	int	r/w, save	Adjust the phase between modulated irig in and irig out, in ns; range is around +/- 100μs.
hw.mcode	uint	r/w, save	Type of installed DIMMs. 0=128M per DIMM, 1=256M per DIMM, ...
hw.memphase	uint	r/w, save	
hw.memflags	int	r/w, save	
hw.adphase	uint	r/w, save	
hw.adcflags	uint	r/w, save	
hw.dcphase	uint	r/w, save	

4.11.4 Touch Panel Calibration

Name	Type	Access	Description
`hw.touchdx`	int	r/w, save	delta x.
`hw.touchdy`	int	r/w, save	delta y.
`hw.touchsx`	int	r/w, save	scale x.
`hw.touchsy`	int	r/w, save	scale y.

4.11.5 Calibrations

Name	Type	Access	Description
`hw.igain`	uint	r/w, save	PRNU correction average gain. A value of 1024 is 1.0X, 1536 is 1.5X, 2048 is 2.0X .
`hw.colorcal`	string	r/w, save	Camera color and white balance calibration matrices.
`hw.sntempcal`	unit	r/w, save	Sensor temperature calibration.

If Z is the raw value for zero degrees C and S is the slope of the temperature sensor in degrees C / raw units, `sntempcal = Z + floor(S * 1000) * 10000`

4.12 Video Structure

4.12.1 General Video Output Controls

Name	Type	Access	Description
`video.system`	uint	r/w, save	Video format code.
`video.output`	uint	r/w, save	Sets video output mode (camera-dependent).
`video.fields`	uint	r/w, save	Not used
`video.widescreen`	float	r/w, save	Set anamorphic desqueeze ratio for some cameras and video modes. A value betwee 0 and 1.0 disables desqueeze.
`video.genlock`	uint	r/w, save	1 = Enable genlock. 3= Enable genlock and reading the VITC (linear time code) from the video return input.
`video.vfmode`	uint	r/w	Viewfinder mode. 0=normal, 1=threshold, 2=zoom, 4=zoom more, 8=show return signal (on some cameras).

4.12.2 On-screen Display Control

Name	Type	Access	Description
`video.osddis`	integer	r/w, save	Turns off the OSD on selected outputs. 1: OSD off on analog and HSDSI1; 2: OSD off on HDSDI2; 3: OSD off on all outputs.

video.osd_ut	boolean	r/w, save	Show UT timestamps instead of local time on the OSD.
video.pax	uint	r/w, save	Production area X size.
video.pay	uint	r/w, save	Production area Y size.
video.paox	int	r/w, save	Production area X offset from center of picture.
video.paoy	int	r/w, save	Production area Y offset

Video.system Values

0	NTSC	2	NTSC color bar pattern
1	PAL	3	PAL color bar pattern
4	720P60	6	720P60 color bars
5	720P50	7	720P50 color bars
12	720P59	14	720P59 color bars
20	1080P30	22	1080P30 color bars
21	1080P25	23	1080P25 color bars
28	1080P29	30	1080P29 color bars
36	1080P24	38	1080P24 color bars
44	1080P23	46	1080P23 color bars
52	1080PSF30	54	1080PSF30 color bars
53	1080PSF25	55	1080PSF25 color bars
60	1080PSF29	62	1080PSF29 color bars
68	1080I30	70	1080I30 color bars
69	1080I25	71	1080I25 color bars
76	1080I29	78	1080I25 color bars
84	1080PSF24	86	1080PSF24 color bars
92	1080PSF23	94	1080PSF23 color bars

4.12.3 Video scaling

Name	Type	Access	Description
video.uzoom	uint	r/w, save	Sets image scaling: 0=zoom to fit, 1=1:1 (or 1:2 for 4k pictures and HD/2k outputs); larger values scale proportionally.
video.vox	uint	r/w, save	Origin of picture window in from the left edge of the video raster.
video.voy	uint	r/w, save	Origin of the picture window from the top edge of the video raster.
video.vow	uint	r/w, save	Width of picture window in video raster pixels.

`video.voh`	uint	r/w, save	Height of picture window in video raster pixels.
`video.vw`	uint	r/w, save	Width of picture window in camera pixels.
`video.vh`	uint	r/w, save	Height of picture window in camera pixels.

4.12.4 Video Image Adjustments

Name	Type	Access	Description
`video.adj.red`	float	r/w, save	Gain of red channel at output of RGB matrix.
`video.adj.green`	float	r/w, save	Gain of green channel.
`video.adj.blue`	float	r/w, save	Gain of blue channel.
`video.adj.toe`	float	r/w, save	Control the gamma curve in the blacks. A value of 1.0 is the default, and matches the gamma curve of previos versions. Decreasing `toe` lifts the blacks, while increasing it compresses them.
`video.adj.gamma`	float	r/w, save	Global gamma for video output. A value of 2.2 makes the camera use a REC709 gamma curve.
`video.adj.rgamma`	float	r/w, save	Difference between red channel gamma and overall gamma for video output.
`video.adj.bgamma`	float	r/w, save	Difference between blue channel gamma and overall gamma for video output.
`video.adj.gain`	float	r/w, save	Global video gain.
`video.adj.offset`	float	r/w, save	Video black level, before gamma LUTs but after white balance.
`video.adj.flare`	float	r/w, save	Flare adjustement, offset before white balance.
`video.adj.hue`	float	r/w, save	REMOVE Hue angle adjustement.
`video.adj.sat`	float	r/w, save	Color saturation adjustement in linear space.
`video.adj.rped`	float	r/w, save	R channel pedestal.
`video.adj.gped`	float	r/w, save	G channel pedestal.
`video.adj.bped`	float	r/w, save	B channel pedestal.
`video.adj.chroma`	float	r/w, save	Chroma gain in YCrCb space.

`video.adj.tone`	string	r/w, save	List of x, y points defining a tone curve for the camera. The list is completed with the implicit points of 0, 0 and 1, 1, and then a tone curve is generated as a cubic spline through the given points. The tone curve is applied to the image after the gamma LUTs.
`video.adj.matrix`	uint	r/w, save	Enable color matrix. When 0, neither the calibration nor the user matrix is applied. However, the white balance component of the calibration matrix is still applied.
`video.adj.log`	uint	r/w, save	Enable log mode for video outputs. A value of zero disables the log mode; Values of 1, 2 etc select different log curves for the outputs driven by the video pipeline A. In log mode, gain, gamma, the pedestals, r/g/b gains, offset and flare are inactive.
`video.adj.sdimin`	uint	r/w, save	Lower clamp value for video outputs (10-bit value). The nominal black level of the outputs does not change with sdimin, and is fixed at 64.
`video.adj.sdimax`	uint	r/w, save	Upper clamp value for video outputs (10-bit value). The image is scaled so that the white point is at sdimax.
`video.adj.cmatrix`	string	r/o	Camera calibration matrix. This matrix is calculated by the camera from the contents of `hw.colorcal` and `wbtemp` and `wbcc`. It results in the output of the camera being converted to the REC709 colorimetry. `cmatrix` is a full RGB matrix, which specifies both a white balance operation and a color correction matrix. The camera separates the matrix into the two operations before applying them.
`video.adj.filter`	string	r/w	Filter calibration matrix. A string with the same format as `hw.colorcal` which describes the effect of a filter to the camera's color correction and white balance.

`video.adj.umatrix`	string	r/w, save	User RGB matrix, applied on the output of the calibration matrix.
`video.adj.wbtemp`	float	r/w, save	White balance temperature. Temperature the camera is white balanced to, in Kelvins. Used to adjust red/blue balance.
`video.adj.wbcc`	float	r/w, save	White balance color compensating index value. Used to adjust green/magenta balance.
`video.adj.wbred`	float	r/w, save	White balance gain (red/green ratio). This variable shows the red/green white balance gain. If set, new `wbtemp` and `wbcc` values will be calculated to match the gain.
`video.adj.wbblue`	float	r/w, save	White balance gain (blue/green ratio). Used together with `wbred`.
`video.adj.mmsat`	string	r/w, save	List of 24 saturation values for multi-matrix color correction. Active when the "mm24" feature is set. Each value is the saturation factor for the respective axis. Set to a null string to disable. Individual axes can be set using the `mmset` command.
`video.adj.mmhue`	string	r/w, save	List of 24 hue corrections (in degrees) for multi-matrix color correction. Active when the "mm24" feature is set. Each value is the hue angle for the respective axis. Set to a null string to disable. Individual axes can be set using the `mmset` command.

The operations specified by `video.adj` are conceptually applied in the following sequence:

1. Offset the raw image by the amount in `flare`;
2. White balance the raw picture using the white balance component of `cmatrix`;
3. Debayer the image;
4. Apply the color correction matrix component of `cmatrix`;
5. Apply the user RGB matrix `umatrix`;

6. Offset the image by the amount in `offset`;
7. Apply the global gain;
8. Apply the per-component gains `red`, `green`, `blue`;
9. Apply the gamma curves; the green channel uses `gamma`, red uses `gamma + rgamma` and blue uses `gamma + bgamma`;
10. Apply the tone curve to each of the red, green, blue channels;
11. Add the pedestals to each color channel, and linearly rescale to keep the white point the same.
12. Convert to YCrCb using REC709 coefficients;
13. Scale the Cr and Cb components by `chroma`.
14. Rotate the Cr and Cb components around the origin in the CrCb plane by `hue` degrees.

4.12.5 Video Playback Control/Status

Name	Type	Access	Description
`video.play.live`	uint	r/w	When 1, the video output is in live mode. When 0, it is in playback mode. Setting this variable to 1 puts the video output in live mode. To start a playback, use a `vplay` command rather than setting this to 0.
`video.play.cine`	uint	r/o	Number of the cine used for playback. Valid only in playback mode.
`video.play.mag`	uint	r/o	Cine in `video.play.cine` is a cinemag cine when this variable is 1. Valid only in playback mode.
`video.play.fn`	int	r/o	Current frame number. Valid only in playback mode.
`video.play.in`	int	r/o	Playback operation in point. Valid only in playback mode.
`video.play.out`	int	r/o	Playback operation out point. Valid only in playback mode.
`video.play.speed`	uint	r/w	Number of fields before the playback frame is updated.

`video.play.step`	int	r/w	Amount by which the frame number is incremented/decremented for each frame update in playback mode. The effective video playback rate is the video format's `pbrate * step / speed`. For backward playback, use negative values for step. For still image, set step to zero.
`video.play.frcount`	uint	r/o	Number of frames remaining from the current playback operation.

All read-only variables in `video.play` except `frcount` are set by the `vplay` command which has started the playback. `frcount` is continuously updated by the camera during playbacks.

4.13 Ethernet Structure

4.13.1 Network settings

Name	Type	Access	Description
`eth.ip`	string	r/w, save	IP address.
`eth.netmask`	string	r/w, save	Network mask.
`eth.broadcast`	string	r/w, save	Broadcast mask.
`eth.gateway`	string	r/w, save	Address of default gateway.
`eth.mtu`	uint	r/w, save	MTU used by the camera.
`eth.xip`	string	r/w, save	IP address for the 10GE.
`eth.xnetmask`	string	r/w, save	Network mask for the 10GE.
`eth.xbroadcast`	string	r/w, save	Broadcast mask for the 10GE.

When `eth.ip` and related fields are set, an alias is created to the camera's primary Ethernet interface, which is configured to the requested settings. The original factory-set address remains active at all times. Setting `eth.ip` to an empty string removes the network interface alias.

The optional 10GE interface on some cameras is not initialised by default, and has no default IP address. It is brought up when `eth.xip` and related fields are set to valid values.

4.14 Cinemag Structure

Name	Type	Access	Description
`mag.state`	uint	r/o	Status of the cinemag subsystem.
`mag.progress`	uint	r/o	Progress indicator for various operations; when the decrementing value reaches zero, the operation is completed.

`mag.protect`	uint	r/o	The attached cinemag has the erase protect switch on.
`mag.size`	uint	r/o	Size of the attached cinemag in kB.
`mag.used`	uint	r/o	Amount of space used in the attached cinemag, in kB.
`mag.takes`	uint	r/o	The number of cines stored to the attached cinemag.
`mag.version`	uint	r/o	Cinemag firmware version.
`mag.id`	string	r/o	Textual description of device attached to the cinemag socket.
`mag.runstop`	int	r/w	Enable direct recording to cinemag.
`mag.type`	uint	r/o	Identification code for device attached to the cinemag socket.

When `mag.runstop` is set, the camera streams images directly to the cinemag or device attached to the cinemag socket whenever it is recording into a cine (a non-preview cine is active). The camera's frame rate is limited accordingly, and the active cine's number of posttrigger frames will be set to 1. If the camera is already recording into a cine when the direct recording mode is set, then the current cine will continue unaffected, and the direct recording will commence starting with the next cine to be recorded.

Mag.type values

`0`	CM1
`100-199`	CM2
`200-999`	Reserved for other cinemag variants
`1000`	Cinestream SR (10GE module using fiber)
`1010`	Cinestream TP (10GE UTP)
`2000`	Cinestream RTO
`3000`	Cinesafe

Mag.state values

`0`	Unplugged
`1`	Power off
`2`	Initialising
`3`	Scanning
`4`	Ready
`5`	Recording
`6`	Playback
`7`	Videoplay
`8`	Erase
`9`	Erase raw

10 Error pending
11 Error
12 Device insertion detected and the device is not a cinemag
13 The device in the cinemag slot is not a cinemag

4.15 Storage Device Structure

Name	Type	Access	Description
cf.state	int	r/o	status of the storage device(see below).
cf.action	uint	r/o	number of active requests on the storage device.
cf.size	uint	r/o	Size of the currently attached storage device, in kB.
cf.used	uint	r/o	Size of the used space on the currently attached storage device, in kB.
cf.progress	uint	r/o	Decrementing counter showing the completion status of the current save operation.
cf.errcode	uint	r/o	Storage device error code.

cf.state values

0 Unplugged
1 Removed
2 Initialising
4 Ready
10 Error pending
11 Error

cf.errcode values

0 no error
1 device mount failed
2 device unmount failed
3 file open failed
4 file control failed
5 file write failed
6 file read failed

4.16 User Settings Sets Structure

Name	Type	Access	Description
`usets.nsets`	uint	r/o	Number of available sets.
`usets.valid#`	boolean	r/o	Status information for set#. When 1, a properly formatted settings block was found at the respective location.

5 Commands

5.1 Get Variable or Structure (get)

Synopsis

```
get <variable_name>
get <structure_name>
get *
```

Arguments

Variable or structure name. The optional "*" argument signifies that all structures are returned.

Description

Read a variable or structure value. When a structure is requested, all it's member's values are returned; substructures are expanded recursively up to the leaf nodes.

Structure reads are atomic with respect to other protocol commands and the machine process. In addition, it is significantly more efficient to read a structure in a single command than to use separate commands for each member.

The response is truncated to the protocol buffer size of 64k.

Response

The response is an ASCII string terminated with an unescaped CRLF. In most cases it will fit on a single line, and contains either a single value or a tagged list. If a structure is requested, the response is broke down into lines, one for each value, each line except the last being terminated with \CRLF. The last line is terminated with an unescaped CRLF.

Errors

```
ERR: expecting  varname
ERR: expecting .
ERR: path too deep
ERR: name <varname> is unknown
ERR: Bad pathlen
```

Examples

```
get info
get info.xver
get c1
```

5.2 Set Variable or Structure (set)

Synopsis

```
set <variable_name> value
set <structure_name>   <tagged_list>
set * <tagged_list>
```

Arguments

Variable or structure name, value or tagged list. The argument "*" signifies that the whole unit structure is set.

A long tagged list can be split across multiple lines with \CR, \LF or \CRLF. The total length of the command (from the command name to the last newline) must not exceed the protocol buffer size of 64k.

Description

Sets a variable or structure value. When a structure is set, only the values of it's members that are explicitly named in the tagged list are updated.

Structure sets are atomic with respect to other protocol commands and the machine process. In addition, it is significantly more efficient to set a structure in a single command than to use separate commands for each member.

Some sets have the side effect of triggering the machine process to update the changed values in the hardware. Since it is particularly important to make sure the values in a structure are consistent at all times, it is better to use aggregate sets.

Response

"Ok!" or an error message.

Errors

```
ERR: expecting  varname
ERR: expecting .
ERR: path too deep
ERR: name <varname> is unknown
ERR: bad pathlen
ERR: missing }
ERR: expecting value
ERR: bad resolution format
ERR: Expecting quoted string
```

Examples

```
set defc.exp 100000
set defc.res 1280x800
```

```
set info.name "my camera"
set defc {res:512x384, rate:1000}
```

The following constructs are equivalent:

```
set defc.rate 1000
set defc {rate: 1000}
set defc.* {rate: 1000}
set * {defc:{rate:{1000}}}
```

5.3 Start Recording in a Cine (`rec`)

Synopsis

```
rec <cine_number>
rec
```

Arguments

The optional `<cine_number>` is the number of a cine from the cine table that is valid.

Description

Start recording in the requested cine. If the specified cine contains a recording, it is silently deleted first. The acquisition parameters are copied from `defc` before the start of recording.

When a `rec` command without arguments is received, if the current active cine is a preview cine and there is another cine in the ready state, the camera begins recording in the latter. If the camera is recording in a non-preview cine or if there is no ready cine available, a `rec` without argument has no effect.

Response

`"Ok!"` or an error message.

Errors

```
ERR: an automatic operation is in progress
ERR: invalid cine number
```

Examples

```
rec
rec 1
```

5.4 Delete a Cine (`del`)

Synopsis

```
del <cine_number>
```

Arguments

The argument `<cine_number>` is the number of a cine in the cine table that is valid.

Description

If the cine is in the stored state, and there is no auto operation running on the cine, the image information recorded in it is discarded, and the cine is returned to the ready state.

Response

`"Ok!"` or an error message.

Errors

```
ERR: missing command args
ERR: missing argvalue for <argname>
ERR: command, "<cmdname>" has no arg named "<argname>"
ERR: invalid cine number
ERR: an automatic operation is in progress
```

Examples

```
del 1
```

5.5 Release a Cine (`rel`)

Synopsis

```
rel <cine_number>
```

Arguments

The argument `<cine_number>` is the number of a cine in the cine table that is valid.

Description

Set the `REU` flag of the cine, allowing it to be deleted automatically by the camera is `auto.acqrestart` is set. This command can be used after the contents of the cine are saved by an external application.

Response

`"Ok!"` or an error message.

Errors

```
ERR: invalid cine number
ERR: an automatic operation is in progress
```

Examples

```
rel 1
```

5.6 Software Trigger (`trig`)

Synopsis

`trig`

Description

The trigger command simulates a hardware trigger. The command does not check whether the currently active cine can be triggered.

After the trigger command (or a hardware trigger) is received, the camera acquires `ptframes` frames in the current cine, marks it as stored (with the `STR` flag) then switches to the next non-preview cine in the table that has the `RDY` flag set. If none is found, it switches to the preview cine.

If the camera is recording directly to a cinemag this command will stop the cinemag recording.

Response

`"Ok!"`

5.7 Get Cine States (`cstats`)

Synopsis

`cstats`

Description

Get Cine States command is used as a more efficient alternative to the multiple commands that would be required to read the status of all cines. If not all the cines in the cine table are valid, this command returns a short list, containing the defined cines present at the start of the table, followed by only one invalid cine.

Response

```
c0 : <status_flag_list>  \
  ...
c# : <status_flag_list>  \
c#+1 : <INV>  \
```

The status flag list is the same that would be returned by a `get c#.state` command (see Section 4.8).

5.8 Start Data Connection (`startdata`)

Synopsis

```
startdata {port:<port_number>}
```

Arguments

The port number to which the camera will attempt to connect.

Description

This command tells the camera software to create a socket connection for transmission of image data and timestamps. The application software should create a socket and accept TCP connections on it, then issue a `startdata` command telling the camera software the port to connect to.

Once the camera succesfully connects, the *data stream* is created. It is used to transfer bulk binary data to the application software. Only one data stream can be open at any given time. A new `startdata` or `attach` command will close the current data stream and open a new one.

The data stream can be terminated by the application software by closing it's end of the socket.

Response

`"Ok!"` when the connection has been successfully established, or an error message.

Errors

```
ERR: missing command args
ERR: missing argvalue for <argname>
ERR: Cannot start data conn
ERR: data transfer disabled
ERR: arg <argname> is mandatory for command <cmdname>
```

Examples

```
startdata {port:15234}
```

5.9 Attach (`attach`)

Synopsis

`attach {port:<port_number>}`

Arguments

The camera will attempt to attach the control connection on which this command is received to a data connection socket identified by the supplied port number.

Description

An application software that has already established a control stream may establish a data stream by connecting a socket to the port 7116 on the camera and instructing the camera to use this new connection as data stream. In order to do that the application software must first connect a socket to the port 7116 on the camera, obtain the TCP port number for this socket and issue the `attach port_number` command.

Once this command completes, the *data stream* is created. It is used to transfer bulk binary data (images and time stamps) to the application software. Only one data stream can be open at any given time. A new `startdata` or `attach` command will close the current data stream and open a new one.

The data stream can be terminated by the application software by closing it's end of the socket.

Response

`"Ok!"` when the connection has been successfully established, or an error message.

Errors

```
ERR: cannot attach on serial lines
ERR: attach failure
```

Examples

`attach {port:15234}` `attach 15234`

5.10 Get Images (img)

Synopsis

```
img {cine:<cine_number>, start:<first_frame>, cnt:<frame_count>
          [, fmt:<format>][, from:<image_source>]}
```

Arguments

`<cine_number>` the cine from which images are taken; A value of `-1` will return live images from the currently active cine.
`<first_frame>` the number of the first frame to send (ignored if `cine_number` is `-1`).
`<frame_count>` the number of frames to send.
`<format>` an optional token or number setting the format in which the images should be sent. The format must be included in `info.imgformats` in order to be valid. The default value is 8.
`<image_source>` an optional token or number indicating the source of data (ignored if `cine_number` is `-1`). Use `ram` or numeric `0` to read from a RAM cine (default). Use `mag` or numeric `1` to read from a cine mag cine.

Description

On the receipt of a `img` command the camera checks the supplied parameters and immediately generates a response. A data transfer request is placed in an internal queue. The actual data transfer (sending the binary image block on the data stream socket) begins after a short latency time. Other `img` or `time` commands can be issued before the data transfer is over.

As `img` and `time` commands are processed, the respective data is sent over the data stream in the order of the requests. The data stream must be set up before `img` and `time` commands can be accepted.

There is some latency in processing the `img` and `time` commands. For optimum peformance when dowloading large data sets, it is recommended that either large blocks of data are requested (e.g. several images) or that new `img` commands are issued while data is read from the data stream.

To get specific images from a cine (as opposed to live images), the cine must be in the stored state. For cameras that have the `earlyimg` feature, it is possible to request images with numbers between `c#.firstframe` and `c#.lastframe` as soon as a cine is triggered.

The image format tokens (`8, 8R`, etc.) have numerical equivalents, listed below. When the camera returns a format, it is generally in the numerical form. The `img` command in `ph16` accepts the format in either token or numerical form.

Token	Number	Description
`8`	8	8 bits per pixel, FPN and PRNU corrected, linear, raw
`8R`	-8	8 bits per pixel, uncorrected, linear, raw
`P16`	272	16 bits per pixel, FPN and PRNU corrected, linear, raw, little-endian. The range of values is 0-65535.
`P16R`	-272	Same as P16 but uncorrected/
`P10`	266	10 bits per pixel packed into 32-bit big-endian words, FPN and PRNU corrected, non-linear, raw. The range of values is 0-65535.

Response

`OK! { cine: <cine_number>, res:<res_x>x<res_y>, fmt: <format>}`

When requesting live images, one cannot always tell from which cine or at which resolution these will be taken. This happens because the active cine can change asynchronously as a result of a hardware event. Also, the resolution of a given cine can change between the moment one reads it with a `get` command and the moment when the `img` command is issued.

When requesting images from a cine stored in RAM, the `fmt` field of the response replicates the corresponding command parameter.

When requesting images from a cine stored to a cine mag, the format of the cine was frozen at save time, and the camera returns this format in the `fmt` field of the response.

The `img` command processing is atomic, so it is guaranteed that the image data sent for a given request comes from the cine and at the resolution given in the response.

Errors

```
ERR: missing command args
ERR: invalid cine number
ERR: cine status invalid
ERR: no cinemag
ERR: start frame outside range
ERR: start+count frame outside range
ERR: count should be > 0
ERR: unsupported image format
ERR: data transfer disabled
```

5.11 Get Images on 10G Ethernet or RTO(ximg)

Synopsis

```
ximg {cine:<cine_number>, start:<first_frame>, cnt:<frame_count>,
     dest:<mac_address>, from:<image_source>}
```

Arguments

`<cine_number>` cine from which images are taken.
`<first_frame>` the number of the first frame to send.
`<frame_count>` the number of frames to send.
`<dest>` a string of 6 hex bytes representing the destination mac address.
`<image_source>` an optional token or number indicating the source of data. Use `ram` or numeric `0` to read from a RAM cine (default). Use `mag` or numeric `1` to read from a cine mag cine.

Description

The operation is similar to the `img` command, except that the images data stream is sent to the 10G Ethernet interface or RTO. When using RTO, a destination mac address must still be supplied, but is not used. The data format for ximg is always `P10`.

5.12 Get Timestamps (time)

Synopsis

```
time {cine:<cine_number>, start:<first_frame>, cnt:<stamp_count>
      [, from:<image_source>]}
```

Arguments

<cine_number> cine from which time stamps are taken. The cine must be in the stored state.
<first_frame> the number of the first frame for which timestamps are requested.
<stamp_count> the number of timestamps to send.
<image_source> an optional token or number indicating the source of data. Use ram or numeric 0 to read from a RAM cine (default). Use mag or numeric 1 to read from a cine mag cine.

Description

On receipt of a time command the camera checks the supplied parameters and immediately generates a response. A data trasfer request is placed in an internal queue. The actual data transfer (sending the binary timestamp block on the data stream socket) begins after a short latency time. Other img or time commands can be issued before the data transfer is over.

As img and time commands are processed, the respective data is sent over the data stream in the order of the requests. The data stream must be setup before img and time commands can be accepted.

There is some latency in processing the img and time commands. For optimum peformance when downloading large data sets, it is recommended that either large blocks of data are requested or that new time commands are issued while data is read from the data stream.

Response

```
OK! {cine:<cine_number>, cnt:<count>, size:<timestamp_size>}
```

The response to the time command contains the information required to correctly interpret the binary data block scheduled to be transmitted on the data stream. <count> is the number of timestamps that are scheduled to be sent, and <timestamp_size> is the size (in bytes) of each time stamp as follows:

cam.tsformat	Size	Description
0	8 bytes	No range data, 16 bits exptime and frac
1	12 bytes	No range data, 32 bit exptime and frac
2	24 bytes	16 bytes of range data, 16 bits exptime and frac.
3	28 bytes	16 bytes of range data, 32 bits exptime and frac.

Errors

```
ERR: missing command args
ERR: missing argvalue for <argname>
ERR: arg <argname> is mandatory for command <cmdname>
ERR: invalid cine number
ERR: cine status invalid
ERR: no cinemag
ERR: data transfer disabled
ERR: count should be > 0
ERR: requested cine not stored
ERR: requested cine has invalid timebuf
ERR: requested frame range is invalid
```

Timestamp Format

Timestamps can have four formats, depending on whether or not the range data input to the camera is enabled, and if the 32 bit exptime and frac extension is used or not. Timestamps mark the moment the shutter closes (end of the exposure) The format of the data as a "C" struct is:

```
struct short_time_stamp{   //cam.tsformat = 0
// time from beginning of the year in 1/100 sec units
    unsigned int   csecs;
// exposure time in us
    unsigned short exptime;
// bits[15..2]: fractions (us to 10000); b[1]:event; b[0]:lock
    unsigned short frac;
};

struct short_time_stamp32{   //cam.tsformat = 1
// time from beginning of the year in 1/100 sec units
    unsigned int   csecs;
// exposure time in us
    unsigned short exptime;
// bits[15..2]: fractions (us to 10000); b[1]:event; b[0]:lock
    unsigned short frac;
// exposure extension up to 32 bits
    unsigned short exptime32;
// fractions extension up to 32 bits
    unsigned short frac32;
};

struct long_time_stamp{   //cam.tsformat = 2
// time from beginning of the year in 1/100 sec units
    unsigned int   csecs;
```

```
// exposure time in us
    unsigned short exptime;
// bits[15..2]: fractions (us to 10000); bit[1]:event; bit[0]:lock
    unsigned short frac;
// first 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d0;
// second 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d1;
// third 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d2;
// fourth 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d3;
};

struct long_time_stamp32{   //cam.tsformat = 3
// time from beginning of the year in 1/100 sec units
    unsigned int   csecs;
// exposure time in us
    unsigned short exptime;
// bits[15..2]: fractions (us to 10000); bit[1]:event; bit[0]:lock
    unsigned short frac;
// exposure extension up to 32 bits
    unsigned short exptime32;
// fractions extension up to 32 bits
    unsigned short frac32;
// first 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d0;
// second 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d1;
// third 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d2;
// fourth 32bits received as rangedata, lsb first, big endian
    unsigned int   range_d3;
};
```

Ints are 32 bits long. Structures are stored in a packed, big endian format.

Following the IRIG time format, time stamps cycle every year. To get the full date, one should add `irig.yearbegin` to the seconds from the timestamp.

Bits 0 and 1 of `timestamp.frac` are flags showing whether the camera time-base is locked to an IRIG timecode source (if `b0:lock = 0`) or the event input of the camera was active when the exposure of the respective frame ended (if `b1:event = 0`).

To reconstruct all the timestamp information, one could use the following expressions:

```
/* calculate the number of seconds since 1970 */
tv_sec = timestamp.csecs / 100 + irig.yearbegin;
/* microsecond offset */
tv_usec = (timestamp.csecs % 100) * 10000 + timestamp.frac >> 2;
/* flags */
locked = (timestamp.frac & 0x01) ? FALSE : TRUE
event_active = (timestamp.frac & 0x02) ? FALSE : TRUE
```

5.13 Show network interface configuration (`ifconfig`)

Synopsis

```
ifconfig
```

Description

Prints out the network interface configuration.

Example response

```
"eth0      Link encap:Ethernet  HWaddr 00:30:64:02:35:4A  \
          inet addr:10.10.10.21  Bcast:10.10.10.255  Mask:255.255.255.0\
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1\
          RX packets:2262018 errors:6 dropped:6 overruns:0 frame:0\
          TX packets:1870630 errors:0 dropped:0 overruns:0 carrier:0\
          collisions:0 txqueuelen:100 \
          RX bytes:1856262819 (1770.2 Mb)  TX bytes:775095837 (739.1 Mb)\
          Interrupt:48 Base address:0xb000 Memory:f6000000-0 \
\
lo        Link encap:Local Loopback  \
          inet addr:127.0.0.1  Mask:255.0.0.0\
          UP LOOPBACK RUNNING  MTU:16436  Metric:1\
          RX packets:420665 errors:0 dropped:0 overruns:0 frame:0\
          TX packets:420665 errors:0 dropped:0 overruns:0 carrier:0\
          collisions:0 txqueuelen:0 \
          RX bytes:156944476 (149.6 Mb)  TX bytes:156944476 (149.6 Mb)\
\
"
```

Errors

```
ERR: ifconfig failed!
```

5.14 Show routing table (`route`)

Synopsis

```
route
```

Description

Prints out the network routing table.

Example response

```
"\
Kernel IP routing table\
Destination Gateway      Genmask       Flags Metric Ref Use Iface\
10.10.10.0  0.0.0.0      255.255.255.0 U     0      0    0  eth0\
127.0.0.0   0.0.0.0      255.0.0.0     U     0      0    0  lo\
0.0.0.0     10.10.10.111 0.0.0.0       UG    0      0    0  eth0\
"
```

Errors

```
ERR: route failed!
```

5.15 Partition cine memory (`partition`)

Synopsis

```
partition { num:<cines> }
```

Arguments

`<cines>` Number of recordable cines to create. The available memory is distributed equally between them.

Description

When running the partition command, all cines are erased. The requested number of new recordable cines is created and the available storage `info.cinemem` is equally divided among them. The new cines are then set to the "ready" state.

The default value for the `cines` argument is 1.

Examples

```
partition {num:3}
partition 2
partition
```

Response

```
OK!
```

The partitioning was performed succesfully.

5.16 Perform white balance (`wbal`)

Synopsis

`wbal`

Arguments

Description

When running the `wbal` command, the camera will calculate the average values of red, green and blue pixels in a small region at the center of the image, and adjust the white balance component of `video.adj.cmatrix`, `video.adj.wbtemp` and `video.adj.wbcc` so that the average color of that region becomes neutral gray.

Response

`OK!`

White balancing started succesfully.

Errors

```
ERR: ref in progress
ERR: cannot do ref
```

5.17 Perform black reference (`bref`)

Synopsis

```
bref [{count:<frame_count>}]
```

Arguments

`<frame_count>` optional argument indicating how many frames are to be accumulated for black reference. The camera limits this argument to the 1..1000 range. Defaults to 8.

Description

When running the `bref` command, the camera will accumulate the following `frame_count` frames and will generate a FPN correction image for the current camera resolution. This correction image will be subsequently used whenever the camera operates at this resolution. For other resolutions the factory supplied black reference is used. During its operation the command decrements the `auto.bref_progress` variable accesible through the `get` command. If the camera has a capping shutter, it will close the shutter before taking frames and reopen it after the `bref` is done.

Response

```
OK!
```

Black reference started succesfully.

Errors

```
ERR: ref in progress
ERR: cannot do ref
```

5.18 PRNU correction update (`wupdate`)

Synopsis

`wupdate`

Arguments

Description

Reload the PRNU correction table from file into the camera RAM. `wupdate` must be used after the PRNU correction file or `hw.igain` has changed. Depending on the camera's maximum image size, this command can take a few seconds to complete.

Response

`OK!`

The update ended.

5.19 Black reference update (`bupdate`)

Synopsis

`bupdate`

Arguments

Description

Reload the FPN correction table from file into the camera RAM. `bupdate` must be used after the FPN correction file has changed. Depending on the camera's maximum image size, this command can take a few seconds to complete.

Response

`OK!`

The update ended.

5.20 Flash erase (`ferase`)

Synopsis

```
ferase
```

Description

This command erase an attached cinemag. Although the command returns immediately, erasing the cinemag takes a sizeable amount of time. During the erase, `mag.progress` is decremented. When the value reaches zero, the erase is complete.

Response

```
OK!
```

Erase started successfully.

Errors

```
ERR: no cinemag
ERR: cinemag not ready
ERR: cinemag is busy
ERR: cinemag failure
ERR: cinemag is erase protected
```

5.21 Flash save (fsave)

Synopsis

```
fsave {cine:<cine>[, firstframe:<firstframe>, lastframe:<lastframe>]}
```

Arguments

`<cine>` Cine to save.
`<firstframe>` Start of the frame range to save. Defaults to the first frame available in cine.
`<lastframe>` End of the frame range to save. Defaults to the last frame available in cine.

Description

This command saves the a frame range within a cine to an attached cinemag. Although the command returns immediately, the save process takes a sizeable amount of time. During the save, `mag.progress` is decremented. When the value reaches zero, the save is complete.

Response

`OK!`

The save started succesfully.

Errors

```
ERR: no cinemag
ERR: cinemag not ready
ERR: cinemag is busy
ERR: cinemag failure
ERR: cinemag is full
ERR: invalid cine number
ERR: no more cine slots
ERR: requested cine not stored
```

5.22 Storage device save (cfsave)

Synopsis

```
cfsave {cine:<cine>[, firstframe:<firstframe>, lastframe:<lastframe>]}
```

Arguments

`<cine>` Cine to save.
`<firstframe>` Start of the frame range to save. Defaults to the first frame available in cine.
`<lastframe>` End of the frame range to save. Defaults to the last frame available in cine.

Description

This command saves the requested frame range within a cine, as a `.cine` file, to an attached storage device. The filename is generated automatically. Although the command returns immediately, the save process takes a sizeable amount of time. During the save, `auto.progress` is decremented. When the value reaches zero, the save is complete.

Response

`OK!`

The save started succesfully.

Errors

```
ERR: no storage device attched
ERR: storage device is busy
ERR: cannot mount volume
ERR: too many active TCP clients
ERR: could not open/create file
ERR: requested cine not stored
ERR: invalid cine number
```

5.23 Retrieve internal camera log (`tail`)

Synopsis

```
tail
```

Description

The camera logs debugging messages into an internal circular buffer. This command allows the retrieval of "tail" end of the buffer.

Response

```
"Ok!{\\r\\n debug messages}\\r\\n"
```

Examples

```
tail
```

5.24 Play frames on video output (vplay)

Synopsis

```
vplay { [cine:<cine>,] [firstframe:<firstframe>,]
        [lastframe:<lastframe>,] [in:<in_point>,]
        [out:<out_point>,] [fn:<frame>,]
        [step:<step>,] [speed:<speed>,] [from:<from>,] }
```

Arguments

<cine> Optional argument selecting a storedcine as the source of frames. If no cine is supplied then vplay will keep playing from the cine that is currently playing. If no cine is playing, the first stored cine is selected.

<firstframe> Alternate name for in.

<lastframe> Alternate name for out.

<in> The start of the frame range to play. Defaults to the c#.in.

<out> The end of the frame range to play. Defaults to the c#.out.

<fn> Frame number the playback is cued to. If fn is not specified, then the current frame of a running playback is maintained. When playback starts in a new cine, fn defaults to the in point if step is positive or zero, or to the out point is step is negative.

<step> Speed and direction control of playback. When step is positive, the camera plays forward. When negative, it will play in reverse. When step is zero, the playback is frozen on the current frame. Larger values of step result in faster playbacks (see also speed). When step is not specified, the playback will use the value from video.play.step.

<speed> Speed control of playback, together with step. Larger values of speed result in a slower playback. The average playback speed is pbrate * step / speed, where pbrate is the natural playback rate of the current video format (field rate for interlaced formats or frame rate for progressive or PsF formats). Defaults to video.play.speed.

<from> Qualifier for the cine number. Valid values are ram and mag. Defaults to ram when cine is specified, or to the value that will maintain the current playing cine if not.

Description

Depending on what arguments are given, `vplay` can be used to initiate a video playback, change the playback speed and direction, or cue a certain frame.

Playback is smoothest if the minimum number of arguments are supplied while a playback is in progress. In particular, `fn` should not be given unless a jump in the play location is desired.

A typical playback session may look like this:

```
vplay {cine:1, step:1, speed:1}   //start playing new cine
//poll video.play to follow the current position
vplay {step:2, speed:1}           //go to 2X speed
vplay {step:0}                    //pause on current frame
vplay {fn: 100, step:0}           //freeze-frame on frame #100
vplay {step:3, speed:2}           //shuttle forward from #100
                                  //at 1.5X speed.
vplay {in:-1000, out:200}         //change playback limits
//poll video.play; when video.play.fn = video.play.out
//the camera is at the end of the playback.
vplay {step:-3, speed:2}          //play backwards at 1.5X speed.
set video.play.live 1             //finish playback, go live.
```

Response

`OK!`

The play started succesfully.

Errors

```
ERR: no stored cines
ERR: invalid cine number
ERR: requested cine not stored
```

5.25 Check for variable changes (`clean`)

Synopsis

```
clean
```

Description

Check if protocol variables have been changed by a command received on a connection other than the current connection or by the on-camera controls. If there were changed since the last time a `clean` command was sent, return an error. This command can be used to synchronise multiple applications controlling a single camera.

Response

```
OK!
```

No changes occured since the last `clean`.

Errors

```
ERR: not clean
```

5.26 Enable status change notifications (`notify`)

Synopsis

```
notify <event_mask>
```

Description

Enable asynchronous event notification on this command socket. For each command socket, the camera maintains an event mask, which is initialised to all zeros when the command socket is created, and can be changed by the `notify` command.

Event notifications are generated by the camera code as various events (such as cine status changes) happen. Each event has an event class bit assigned. If the corresponding bit in the event mask for a given connection is set, an asynchronous notification will be sent on that connection.

The notification consists of a one-line message of the form:

```
@<event_name>@\CRLF
```

The notifications are generally sent in-between protocol responses, but it is possible that the event message would appear within a response in some cases.

Event Name	Class	Description
`startaq`	1	A new cine has started recording.
`trig`	1	The current cine was triggered.
`stored`	1	The current cine finished recording and went to the stored state.

Response

```
OK!
```

5.27 Save factory defaults (`isave`)

Synopsis

```
isave
```

Description

Save the complete unit structure to a backup file.

Response

```
Ok!
```

Errors

```
ERR: factory defaults save failed
```

Examples

```
isave
```

5.28 Load factory defaults (`iload`)

Synopsis

```
iload
```

Description

Load factory defaults as saved by `isave`.

Response

```
Ok!
```

Errors

```
ERR: factory defaults load failed
```

Examples

```
iload
```

5.29 Save user settings (usave)

Synopsis

```
usave { slot: <number>[, name: <string>] }
```

Description

Save the current camera settings to a specified settings file. `usave` only saves the settings that are not specific to a particular camera, so the settings block can be copied to a different camera of the same type. This differs from the `isave` command, which saves all the parameters of the camera. The optional name parameter will set the name of the given settings set.

Response

```
Ok!
```

Errors

```
ERR: user slot error
```

Examples

```
usave 2
usave {slot: 2}
```

5.30 Load user settings (uload)

Synopsis

```
uload { slot: <number> }
```

Description

Load user settings from the specific settings file. The settings must have been created by a camera of the same type in order to be accepted for reading.

Loading a set of settings will erase all stored cines from memory and place the camera in PTR mode.

Response

```
Ok
```

Errors

```
ERR: user slot error
```

Examples

```
uload 1
uload {slot: 1}
```

5.31 Erase user settings (`uerase`)

Synopsis

```
uerase { slot: <number> }
```

Description

Erase an user settings file. The argument is the slot number to be erased.

Response

```
Ok!
```

Errors

```
ERR: user slot error
```

Examples

```
uerase 2
```

5.32 List user Settings (`uls`)

Synopsis

```
uls
```

Description

List numbers and names of the valid user settings slots.

Example Response

```
{1, "setup1",\
4, "four" }
```

Examples

```
uls
```

5.33 Debug console mode (`console`)

Synopsis

```
console { level: <number> }
```

Description

The command enters/exits the debug console mode on the current protocol connection. It accepts a numeric argument representing the debug level. During debug console mode the debugging messages generated by the camera (see the `tail` command) are also printed out on the current connection if their level is lower than the requested debug level.

Level	Description
-1	Error messages.
0	Informational messages about major events.
1..5	Other messages (verbosity increases with level).

Any value lower than `-1` can be used to exit the debug console mode.

Any protocol command error will abort the printout of currently pending message strings and will print it's error message instead.

This command is intended to be used only on human initiated telnet connections, since, while in the debug console mode, the protocol covention of getting one answer for each command is violated.

Response

```
Ok!
```

Examples

```
console 3
console -2
```

5.34 Set lens aperture (`fstop`)

Synopsis

```
fstop { value: <aperture>}
fstop
```

Description

Set the lens aperture to the specified value. Aperture values are floating point numbers. When an aperture of zero is requested, the lens will go wide open.

The `fstop` command without argument returns the current aperture setting of the lens.

Response

```
Ok!
Ok! {fstop: value}
```

Errors

```
ERR: no lens
ERR: bad arg
```

Examples

```
fstop
fstop 2.8
fstop 0
```

5.35 Move focus (`focus`)

Synopsis

```
focus { value:<focus change>}
focus
```

Description

Request the lens to move the focus ring by the specified number of incremental units (ticks). All focus moves are relative to the current focus position. Lenses may have a hysteresis of a couple of ticks when changing focus direction. Focus units (ticks) are of arbitrary size, and uncalibrated. The full range of focus of most lenses is of the order of 1000-3000 ticks. The command without argument returns the current focus state. Possible values are:

`focus state`	Description
`ok`	Last focus change operation completed succesfully
`manual`	Last focus change failed. Lens is set to manual focus
`limit`	The focus adjustment has hit a mechanical stop during the last move command.
`progress`	Focus change operation is in progress.
`unknown`	No focus change operation was requested since powerup

Response

```
"Ok! {focus: limit}
"Ok! {focus: manual}
"Ok! {focus: ok}
"Ok! {focus: progress}
"Ok! {focus: unknown}
"Ok!
```

Errors

```
ERR: no lens
ERR: bad arg
```

Examples

```
focus
focus 100
```

5.36 Issue Lens Mount Command (`lens`)

Synopsis

```
lens lens_mount_command_string
```

Description

The quoted string argument is unquoted and passed directly to the lens mount, for debugging and test purposes. An eventual response string returned by the lens mount is available through the `tail` command.

Response

```
Ok!
```

Errors

```
ERR: bad arg
```

Examples

```
lens "#sx"
lens "#sma i 0"
lens "#sma r"
```

5.37 Change serial line baud rate (`baud`)

Synopsis

```
baud { rate: <number> }
```

Description

The command changes the serial line baud rate for the current session. It accepts a numeric argument representing the new baud rate. The valid baud rate values are 38400, 115200 and 230400. The power-on baud rate is always 38400 bps.

If multiple serial lines are present, a `baud` command issued over an Ethernet connection will change the speed for all serial lines, while a `baud` command issued over a serial line will change the speed for that serial line only.

Response

```
Ok!
```

Errors

```
ERR: only 38400, 115200 and 230400 supported
```

Examples

```
baud 115200
```

5.38 Camera calibration (`calib`)

Synopsis

```
calib { what: <operation> }
```

Description

The command performs basic camera calibration routines as follows:

`Operation`	Description
`1`	Perform black reference.
`2`	Save black reference.
`3`	Save white reference.
`4`	Save white reference.
`5`	Disable corrections.
`6`	Enable corrections.

Response

```
Ok!
```

Errors

```
Err: 1=bref, 2= bsave, 3=wref, 4=wsave
```

Examples

```
calib 2
```

5.39 System monitor (sysmon)

Synopsis

```
sysmon
sysmon { dev:<devnum>, content:<data_string>
```

Description

Without arguments, sysmon searches for all one-wire devices in the camera and returns their serial numbers and saved data. Also, it returns available system monitor information, as voltages and temperatures.

When given a device number and data string, the command will write the data string to te given one-wire device.

Response

The response of the command is a list of strings. Each string contains a system monitor item: either a one-wire device data, or monitor information for one board. The first token of each string describes the type of the data item. The rest of the string is item-dependent.

An example of response is:

```
{\
"dev[0]: 56 00 00 01 30 CE CC 2D 'VRO07295MIROCPU004REVT2'",\
"dev[1]: A3 00 00 01 30 C8 7E 2D 'VRO07295MIROMB004REVT2'",\
"dev[2]: 3A 00 00 01 30 D7 85 2D 'VRO07295MIROSN004REVT2'",\
"dev[3]: E2 00 00 01 30 D5 95 2D ''"\
}
```

Errors

```
ERR: one_wire read error
ERR: no OW devices
ERR: missing content string
ERR: missing end of content string
ERR: one_wire error
```

Examples

```
sysmon
sysmon 2 "VRO07295MIROCPU004REVT2"
```

5.40 Generate test image (`testimg`)

Synopsis

```
testimg { type: <image_type> }
```

Description

The command instructs the camera to generate one of several test images, or to function normally.

image_type	Description
0	the camera operates normally.
1	the camera generates a test image containing a circle.
2	the camera generates a test image loaded from a file.
3	the camera generates a test image containing a triangle.
4	the camera generates a test image containing a gray level.

Response

```
Ok!
```

Examples

```
testimage 1
```

5.41 Set Real Time Clock (`setrtc`)

Synopsis

```
setrtc { value: <seconds> }
```

Description

Sets the real time clock within the camera to the specified time, which is seconds from the beginning of 1970 UTC.

Response

```
Ok!
```

Examples

```
setrtc 1122334455
```

5.42 List files on storage device (`cfls`)

Synopsis

```
cfls
```

Description

Lists all files existing on the attached storage device.

Response

```
{"Midi 2_1123_0.cine", 39688208, "2011-8-29 14:14:58", \
"Midi 2_1123_1.cine", 45678900, "2011-8-29 15:24:32"}
```

Examples

```
cfls
```

5.43 Remove file from storage device (`cfrm`)

Synopsis

```
cfrm { filename: <name> }
```

Description

Removes the file `name` from the attached storage device.

Response

```
Ok!
```

Examples

```
cfrm "Midi 2_1123_0.cine"
```

5.44 Format storage device (`cfformat`)

Synopsis

```
cfformat
```

Description

Formats the attached storage device. This command returns Ok after the format is complete, which can take several tens of seconds.

Response

```
Ok!
```

Errors

```
ERR: cannot format
```

5.45 Read file data from storage device (cfread)

Synopsis

```
cfread {filename:<name>, offset:<offs>, count:<num_bytes>}
```

Arguments

<name> name of the file to read.
<offset> 64 bit value specifying the offset from the start of the file.
<count> 32 bit value specifying the amount of bytes to be read.

Description

This command tries to read an amount of num_bytes bytes starting at the offset offs from the file name on the external storage device and returns the number of bytes really read.
The file binary data arrives to it's requestor via the data stream.
A return value of 0 signals EOF.

Response

```
OK! {count: 1048576}
```

One megabyte of file data will arrive through the data stream.

Errors

```
ERR: startdata required
ERR: no buffers
ERR: CF not ready
ERR: cinemag failure
ERR: cannot open file
ERR: offset too big
ERR: read error
ERR: too many pending requests
```

5.46 Use color preset (preset)

Synopsis

```
preset {[matrix:<mat>,] [tone:<ton>,] [filter:<filt>]}
```

Arguments

<mat> Optional value in the 1..4 range specifying the matrix preset to be used.
<ton> Optional value in the 1..4 range specifying the tone preset to be used.
<filt> Optional value in the 1..4 range specifying the filter preset to be used.

Description

This command copies the selected preset values in their video.adj counterparts.

Errors

ERR: bad arg

5.47 Set multi-matrix axis (mmset)

Synopsis

```
mmset {axis: <angle>, sat: <saturation>, hue: <hue angle> }
```

Description

Set the saturation/hue values for the axis closest to the specified angle into the `video.adj.mmsat` and `video.adj.mmhue`.

Response

```
Ok!
```

Examples

```
mmset {axis:180, sat:1.1, hue:-7.5}
```

6 Discovery Protocol

Phantom cameras present on a network can be detected using the following *discovery protocol*:

A UDP packet containing the eight-character string `phantom?` is broadcast to port `7380`. Every active camera on the network responds with a UDP packet containing a character string `PH16 <port> <hwver> <serial>`. The `<port>` field is the port number (normally 7115) on which the camera accepts control stream connections.`<hwver>` is the hardware version of the camera as in `info.hwver`, and `<serial>` is the serial number of the camera as in `info.serial`.

7 Example camera resolution and metadata settings for various picture formats on the Phantom 4k

Format	defc.res	w	h	ow	oh	cr	rs	ws
4096x2304 (native 16:9)	4096x2304					0	0	0
3840x2160 (overscan 16:9)	4096x2304	3840	2160	3840	2160	1	0	0
2752x2304 (anam. 2.0)	4096x2304	2752	2304	2752	2304	1	0	2.0
1920x1080 (scaled 16:9)	4096x2304	4096	2304	1920	1080	0	1	0
4096x2160 (native DCI)	4096x2160					0	0	0
4096x2216 (native 1.85:1)	4096x2216					0	0	0
4096x1712 (native 2.39:1)	4096x1712					0	0	0
3840x2160 (cropped 16:9)	4096x2160	3840	2160	3840	2160	1	0	0
2048x1080 (scaled DCI)	4096x2160	4096	2160	2048	1080	0	1	0
2048x1536 (native 4:3)	2048x1536					0	0	0
2048x1152 (native 4:3)	2048x1152					0	0	0
1920x1080 (overscan 16:9)	2048x1152	1920	1920	1920	1080	1	0	0
2048x1080 (native DCI)	2048x1080					0	0	0
1920x1080 (crop 16:9)	2048x1080	1920	1080	1920	1080	1	0	0
1280x720 (scaled 16:9)	2048x1152	2048	1152	1280	720	0	1	0
1280x720 (crop 16:9)	2048x720	1280	720	1280	720	1	0	0

8 Pin Functions for Fischer 12-pin Capture Ports

Pin	Color	Mode	Old MIRO	Midi/SRC	FLEX 4k	JB2 Uplink
1			GND (signal)			
2			GND (power)		GND (signal)	
3			GND (power)		GND (signal)	
4	Red		Trigger in	Trigger io	Trigger in	Trigger in
5	Green	AUX2=0	Ready	Ready	Ready	Ready
		AUX2=1	Ready	Strobe	Strobe	N/A
		AUX2=2	Ready	N/A	AES out	N/A
6	Black	AUX1=0	Fsync io	Strobe	Strobe	N/A
		AUX1=1	Fsync io	Event	Event	Event in
		AUX1=2	Fsync io	Mem gate	Mem gate	Mem gate in
		AUX1=3	Fsync io	Fsync io	Fsync io	Fsync in
7	White		Irig in			
8			GND reference for pin 9			
9	Yellow		Video out	Video out	AES in	N/C
10			Power	Power	N/C	N/C
11			Power	Power (acc)	N/C	N/C
12	Blue	IRIGEN=0	Strobe	Irig out	Irig out	Irig out
		IRIGEN=1	Irig out	Irig out	Irig out	Irig out

9 Revision Log

1.0 First version of the document;
1.1 Add edr feature string, `hw.sntempcal`
1.2 Add notify feature string, make video.play.fn readonly, new arguments to vplay.
1.4 Changed the contents of the cf struct. Added irig.gps, c.meta.gps. Added commands: cfls, cfrm, cfformat, cfread, testimg, setrtc.
1.5 Add description for the preset struct and children, video.adj.filter, preset command.
1.6 Add aux2mode, option 3 for aux1mode; new feature string descriptions; added video.adj.fields: wbred, wbblue, sdimin, sdimax, toe, loga, logb.
1.7 Add 12-pin capture connector table.
1.8 Fixed units of `cam.frdelay` to ns.
1.9 Added `adj.mmsat`, `adj.mmhue`, the `roi` and `mmset` commands and the `mm24` and `roi` feature strings.
1.10 Removed `roi`; Added `video.widescreen`, log mode description in the `cine.adj` section, example metadata settings for various formats; clarified definition of `meta.ox`, `meta.oy`.
2.0 Small corrections in the formats table, switch to a single `log` setting instead of `loga` and `logb`.
2.1 Add `wide` feature string.
2.2 Fixed formatting at feature string table.
2.3 Add name parameter to `usave`, new value for `video.genlock`, the description of the `uls` command.