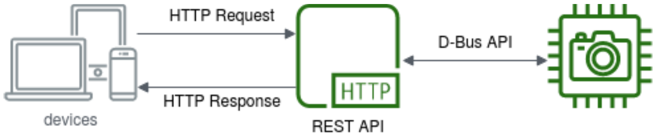
Overview

Control of the Chronos camera is provided as a REST API, which is a type of web API, involving requests and responses, not too unlike visiting a web page. You make a request to a resource stored stored on a server, and the server responds with the requested information. The protocol used to transport the data is HTTP. "REST" stands for Representational State Transfer.



The Chronos API provides access to the camera configuration, settings and related data describing the camera's hardware and available features. The base address of the Chronos API is `http://192.168.12.1/control` when accessing the camera via its USB interface. This API provides a set of endpoints, each with its own unique path.

Methods

API methods are procedures that may start a procedure, or change the camera state. Since these operations do not fit well into the REST model, they are performed using the HTTP POST method, with their arguments provided in JSON format as the HTTP POST body.

describe

The `describe` method is accessible by the `/control/describe` endpoint, and returns a description of the available parameters and methods that can be accessed via the Chronos API. This method is used to generate most of the reference information on this page.

```
user@example.com:~$ curl http://192.168.12.1/control/describe
{
    "cameraMemoryGB": {
        "type": "d",
        "get": true,
        "set": false,
        "notifies": false,
        "doc": "int: Amount of video memory attached to the FPGA in GiB"
    }
    ...
}
```

Member	Description
`type`	D-Bus type signature for the parameter's value.
`get`	`true` when the parameter can be retrieved using the `get` method
`set`	`true` when the parameter can be changed using the `set` method
`notifies`	`true` when changes to the parameter are reported using the `notify` event
`doc`	User documentation string, explaining the parameter's meaning and function

availableCalls

The `availableCalls` method is accessible by the `/control/availableCalls` endpoint.This method gets a list of the methds that can be called via the API.

This method returns a dictionary with an entry for each method that can be called via the API. Each entry will include a `brief` string that summarizes the purpose of the API method. Optionally, the entries may also contain a `description` with a more extensive detail, as well as `args` and `returns` dictionaries that list the parameters that the method accepts, and any values that the method returns.

The `availableCalls` method returns a dictionary with the following members:

Return Value	Type	Description
`calls`	dict	A dictionary describing each method that is callable by the API.

availableKeys

The `availableKeys` method is accessible by the `/control/availableKeys` endpoint.This method gets a list of the parameters available in the API.

This method returns a dictionary with an entry for each parameter that can be accessed via the API. Each entry will describe the `type` of the parameter as a D-Bus signature, a `doc` string that describes the function of the parameter, as well `get`, `set`, and `notify` flags that indicate whether the parameter can is read-only, read-write or generates `notify` events when its value changes.

The dictionary for each key may also include additional details depending on the type of the parameter. String parameters describing an enumerated type, may include an `enum` dictionary which maps each of the acceptable values to a brief docstring describing what that value does.

Dictionary types may include an `args` dictionary describing the each member of the dictionary does when it is set in the API, or they may include a `returns` dictionary describing what each dictionary member means when it is returned by the API.

Each key may also include a `description` member, which provides a detailed multi-line documentation string. This is intended to provide more detail than may be available in the single-line `doc`.

The `availableKeys` method returns a dictionary with the following members:

Return Value	Type	Description
`keys`	dict	A dictionary describing each parameter in the API.

clearCalibration

The `clearCalibration` method is accessible by the `/control/clearCalibration` endpoint.This method removes user calibration data, returning the camera to its factory state.

When called with no arguments, this removes only the user calibration, allowing the camera to return to its factory new state. The caller may also specify the removal of factory calibration data, though this is not recommended unless the user has made a backup of their calibration data first.

The `clearCalibration` method accepts the following arguments:

Argument	Type	Description
`factory`	bool, optional	Also remove factory calibration data. (default: false)

exportCalData

The `exportCalData` method is accessible by the `/control/exportCalData` endpoint.This method generates factory calibration samples and saves them to external storage

This method iterates through the image sensor's internal calibration modes and generates factory calibration sample data to be processed externally. The calibration data will be saved to a USB thumb drive, typically mounted at /media/sda1.

After external processing of the calibration samples is complete, the resulting calibration data can be imported to the camera using the `importCalData` method.

flushRecording

The `flushRecording` method is accessible by the `/control/flushRecording` endpoint.This method flushes recoreded video data from memory.

Normally when recording video, the camera will overwrite video data only as needed to make room for new data from the the image sensor. This method discards all video data from the video memory so that the user can start fresh on their next recording.

get

The `get` method is accessible by the `/control/get` endpoint.This method retrieves parameter values from the API.

The resulting dictionary will contain an element for each parameter that was successfully read from the API. If any parameters could not be read, they will be included in an `error` dictionary giving the reasons that they could not be retrieved.

The `get` method accepts the following arguments:

Argument	Type	Description
`*names`	string	list of parameter names to rerieve from the API.

getResolutionTimingLimits

The `getResolutionTimingLimits` method is accessible by the `/control/getResolutionTimingLimits` endpoint.This method tests the camera ability to support a desired resolution and framerate.

This method checks the sensor's ability to operate at the desired resolution parameters and, if successful, reports on some of the parameters that would apply if that resolution was configured. Otherwise, this method will generate an error to indicate that the resolution setting is not supported by the image sensor.

The `getResolutionTimingLimits` method accepts the following arguments:

Argument	Type	Description
`bitDepth`	int, optional	Desired pixel bit depth to use for image readout. (default: image sensor maximum)
`hOffset`	int, optional	Horizontal offset of the image from the right edge of the image sensor. (default: center the image horizontally)
`hRes`	int	Horizontal image resolution, in pixels.
`minFrameTime`	float, optional	Minimum time period, in seconds between frames, that the imager sensor will operate at. (default: image sensor minimum)
`vOffset`	int, optional	Vertical offset of the image from the top edge of the image sensor. (default: center the image vertically)
`vRes`	int	Vertical image resolution, in pixels.

The `getResolutionTimingLimits` method returns a dictionary with the following members:

Return Value	Type	Description
`cameraMaxFrames`	int	The maximum number of frames that the camera can save at this resolution and framerate setting.
`exposureMax`	int	The maximmum exposure period in nanoseconds, that the image sensor can expose a frame for if framePeriod was set equal to minFramePeriod.
`exposureMin`	int	The minimum exposure period in nanoseconds that the image sensor can exposure a frame for.
`minFramePeriod`	int	The minimum frame period, in nanoseconds between frames, that the image sensor can operate at.

importCalData

The `importCalData` method is accessible by the `/control/importCalData` endpoint.This method imports calibration data that was generated off-camera.

This method looks for any calibration data present on a USB thumb drive, typically mounted at /media/sda1, and copies the calibration data to the camera's internal filesystem for later use.

This method is used during factory calibration to import calibration data that the camera is not capable of generating on its own. Typically the camera will be connected to a test jig to stimulate the camera, with data being acquired using the `exportCalData` method.

reboot

The `reboot` method is accessible by the `/control/reboot` endpoint.This method restarts the control API and/or the camera.

This method allows the user to restart their camera software, and optionally perform a full power cycle and/or return to factory default settings at the same time.

The `reboot` method accepts the following arguments:

Argument	Type	Description
`power`	boolean, optional	When true, the camera will perform a full power cycle.
`reload`	boolean, optional	When true, the control API and user interfaces will restart themeselves (default: true).

Argument	Type	Description
`settings`	boolean, optional	When true, the user and API settings are removed during the reboot, returning the camera to its factory default state.

set

The `set` method is accessible by the `/control/set` endpoint.This method sets parameter values in the API.

The resulting dictionary will contain an element for each paramer that was successfully set in the API. If any parameters could not be set, they will be included in an `error` dictionary given the reason that they could not be set. Typically this is either because the value given was not valid for the parameter, or the parameter did not exist.

The `set` method accepts the following arguments:

Argument	Type	Description
`**values`	dict	A dictionary naming each of the parameters to update, and the to which they should be set.

startCalibration

The `startCalibration` method is accessible by the `/control/startCalibration` endpoint.This method begin one or more calibration procedures at the current settings.

Black calibration takes a sequence of images with the lens cap or shutter closed and averages them to find the black level of each pixel on the image sensor. This value is then be subtracted during playback to correct for image offset defects.

Analog calibration consists of any automated image sensor calibration that can be performed quickly and autonomously without any setup from the user (eg: no closing of the aperture or calibration jigs).

Factory calibration algorithms may require special test equipment or setups. Factory calibration also implies that calibration data will be saved, and that conflicting user calibration data will be removed.

The `startCalibration` method accepts the following arguments:

Argument	Type	Description
`analogCal`	bool, optional	Perform autonomous analog calibration of the image sensor. (default: false)
`blackCal`	bool, optional	Perform a full black calibration assuming the user has closed the aperture or lens cap. (default: false)
`factory`	bool, optional	Whether factory calibration algorithms should be performed. (default: false)
`saveCal`	bool, optional	Whether the results of calibration should be saved to the filesystem for later use. (default: false)
`zeroTimeBlackCal`	bool, optional	Perform a fast black calibration by reducing the exposure time and aperture to their minimum values. (default: false)

This method starts an asynchronous process that changes the camera's state and executes in the background. The results of the `startCalibration` method will be returned to the user in the `complete` event, with a `method` equal to `startCalibration`.

startFilesave

The `startFilesave` method is accessible by the `/control/startFilesave` endpoint.This method saves a region of recorded video to external storage.

Upon calling this method, the video system will switch to the `filesave` state and begin encoding video data to the output `device`. During this procedure, the `playbackStart`, `playbackPosition` and `playbackLength` parameters will be updated to track the progress of the filesave.

When the filesave is completed, the video system will exit the `filesave` state, and revert back to whichever state it was in when the `startFilesave` method was called.

The `startFilesave` method accepts the following arguments:

Argument	Type	Description
`bitrate`	int, optional	For compressed formats, this sets the desired bitrate of the encoded file in bits per second (0.25 bits per pixel per second).
`device`	string	Name of the external storage device where video should be saved.
`filename`	string, optional	Name to give to the video file (or directory for TIFF and DNG formats). When omitted, a filename is generated using the current date and time.
`format`	string	Enumerate the output video format.
`framerate`	int, optional	For formats with a media container (such as MPEG-4), this determines the framerate of the encoded media file (default: 60 frames per second).
`length`	int, optional	The number of frames of video that should be saved (default: all frames).
`start`	int, optional	The frame number in recorded video where the saved video begin (default: 0).

startLivedisplay

The `startLivedisplay` method is accessible by the `/control/startLivedisplay` endpoint.This method switches the video system into live display mode.

When in live display mode, the camera will replay the active video data being acquired from the image sensor onto the LCD screen, HDMI port and its RTSP stream. The video stream will monitor for changes in the video geometry, or hotplug events and may restart and reconfigure itself as necessary to keep the video data flowing. The show must go on.

Any video properties that relate to video playback rate and position have no meaning or effect when in this state.

startPlayback

The `startPlayback` method is accessible by the `/control/startPlayback` endpoint.This method switches the video system into playback mode, or sets the playback position and rate.

When in playback mode, the camera will replay the captured video on the LCD, HDMI port and its RTSP stream. The user may configure the starting frame number and the rate at which video is replayed.

The actual video stream replayed by the camera is fixed at either 30 or 60fps, the camera will either skip or duplicate frames to achieve the requested framerate. For example, setting the `framerate` to 120fps will typically play every 2nd frame at 60fps.

The `framerate` can be either positive for forward playback, or negative to rewind backwards through video. A value of zero will effectively pause the video on the current frame.

The `startPlayback` method accepts the following arguments:

Argument	Type	Description
`framerate`	int	The rate, in frames per second, at which video should advance through the playback memory.
`loopcount`	int, optional	The number of frames, after which the video system should return back to `position` and continue playback. This allows the user to select a subset of the video to play.
`position`	int	The starting frame number from which video should play.

startRecording

The `startRecording` method is accessible by the `/control/startRecording` endpoint.This method program the recording sequencer and start recording.

The `startRecording` method accepts the following arguments:

Argument	Type	Description
`recMode`	RecModes, optional	Override the current `recMode` property when starting the recording.

This method starts an asynchronous process that changes the camera's state and executes in the background. The results of the `startRecording` method will be returned to the user in the `complete` event, with a `method` equal to `startRecording`.

startWhiteBalance

The `startWhiteBalance` method is accessible by the `/control/startWhiteBalance` endpoint.This method begin the white balance procedure.

Take a white reference sample from the live video stream, and compute the white balance coefficients for the current lighting conditions. If successful, the results of the white balance calculation will be stored in `wbCustomColor` and `wbTemperature` will be set to 0K.

The `startWhiteBalance` method accepts the following arguments:

Argument	Type	Description
`hStart`	int, optional	Horizontal position at which the white reference should be taken.
`vStart`	int, optional	Veritcal position at which the white reference should be taken.

This method starts an asynchronous process that changes the camera's state and executes in the background. The results of the `startWhiteBalance` method will be returned to the user in the `complete` event, with a `method` equal to `startWhiteBalance`.

stopFilesave

The `stopFilesave` method is accessible by the `/control/stopFilesave` endpoint.This method terminates an ongoing filesave operation

When the video system has started a filesave operation, it can take a very long time to complete denepding on the quanitity of footage being saved, and the speed of media to which it is being written. If operation was started in error, or the user changes their mind, then this method may be used to terminate that operation rather than waiting for it to complete.

It is acceptable to call this method even when no filesave operation is in progress, however, it may result in an otherwise unexpected restart of the video system.

stopRecording

The `stopRecording` method is accessible by the `/control/stopRecording` endpoint.This method terminate a recording if one is in progress.

Events

With server-sent-events it is possible for the camera to send asynchronous notifications when long running operations complete, or parameters change in the API. This is done by pushing events to the web browser.

Using Javascript, a browser can subscribe to the HTML5 Server-Sent-Events stream by creating a new `EventSource` on the `/control/subscribe` endpoint, and then using the `addEventListener` function to receive events.

```
function onNotifyEvent(data) {
  document.getElementById("result").innerText = JSON.parse(data);
}
var evtSource = new EventSource("/control/subscribe");
evtSource.addEventListener("notify", function(event) {onNotifyEvent(event.data);});
```

notify

The `notify` event is generated whenever a mutable parameter in the API changes its value, and the data sent with the event will contain a dictionary of the updated parameter values.

```
user@example.com:~$ curl http://192.168.12.1/control/subscribe
event: notify
data:{
data:   "calSuggested": false,
data:   "state": "analogcal"
data:}
```

complete

The `complete` event is generated whenever an asynchronous procedure has run to completion, and will contain the results of the procedure. If the procedure completed successfully then the data will contain a dictionary with the name of the method the completed, and the new state of the camera. If the procedure completed with an error, then the dictionary will also contain an `error` with the type of error that occured, and optionally a `message` with a human-readable description of the error.

```
user@example.com:~$ curl http://192.168.12.1/control/subscribe
event: complete
data:{
data:   "state": "idle",
data:   "method": "startWhiteBalance",
data:   "error": "SignalClippingError",
data:   "message": "Signal clipping, reference image is too bright for white balance"
data:}
```

Member	Description
`state`	The new state of the camera after completing the asynchronous call
`method`	The name of the asynchronous API call that has completed
`error`	A canonical name for an error that occured during the asynchronus call (optional)
`message`	A human-readable string describing the cause of the error

Parameters

The Chronos API exposese a set of parameters that are accessible using a REST API. Parameters are accessed via standard HTTP requests in JSON format, and where possible the Chronos API uses appropriate verbs for each action:

Verb	Endpoint	Action
GET	`/control/p/{name}`	Retrieve a single parameter by name if the `r` flags is set.
PUT	`/control/p/{name}`	Set the value of a single parameter by name if the `w` flag is set.
POST	`/control/p`	Update a collection parameters together

Parameters will have one or more flags describing the ways in which they can be manipulated using the REST API:

- `r` flag: The parameter's value can be retrieved using the HTTP GET verb.
- `w` flag: The parameter's value can be updated using the HTTP SET verb.
- `n` flag: Changes to the parameter's value will be reported using the `notify` event.

Name	Type	Flags	Description
`backlightEnabled`	`boolean`	`rwn`	True if the LCD on the back of the camera is lit. Can be set to False to dim the screen and save a small amount of power. backlightEnabled => true

Name	Type	Flags	Description
batteryChargeNorma	float	r--	Estimated battery charge, with 0.0 being depleted and 1.0 being fully charged. batteryChargeNormalized => 1
batteryChargePerce	float	r--	Estimated battery charge, with 0% being depleted and 100% being fully charged. batteryChargePercent => 100
batteryCritical	boolean	r-n	True when the battery voltate is critically low and a powerdown is imminent batteryCritical => false
batteryPresent	boolean	r-n	True when the battery is installed, and False when the camera is only running on adaptor power batteryPresent => true
batteryVoltage	float	r--	The voltage that is currently being output by the battery. A fully charged battery outputs between 12V and 12.5V. batteryVoltage => 12.405
calSuggested	boolean	r-n	True when the calibration of the camera needs updating. calSuggested => false
cameraApiVersion	string	r--	Version string of the pychronos module cameraApiVersion => "0.4.0-beta"
cameraDescription	string	rwn	Descriptive string assigned by the user cameraDescription => "Chronos SN:01436"
cameraFpgaVersion	string	r--	Version string of the FPGA bitstream that is currently running cameraFpgaVersion => "3.24"
cameraIdNumber	int	rwn	Unique camera number assigned by the user cameraIdNumber => 0

Name	Type	Flags	Description
`cameraMaxFrames`	`int`	`r-n`	The maximum number of frames the camera's memory can save at the current resolution. cameraMaxFrames => 17470
`cameraMemoryGB`	`float`	`r--`	Amount of video memory attached to the FPGA in GiB cameraMemoryGB => 32
`cameraModel`	`string`	`r--`	Camera model name cameraModel => "CR14-1.0"
`cameraSerial`	`string`	`r--`	Unique camera serial number cameraSerial => "Nicholas!"
`cameraTallyMode`	`string`	`rwn`	Mode in which the recording LEDs should operate. • **off**: All recording LEDs on the camera are turned off. • **auto**: The recording LEDs on the camera are on whenever the `status` property is equal to 'recording'. • **on**: All recording LEDs on the camera are turned on. cameraTallyMode => "auto"

`colorMatrix`	`array[fl`	`rwn`	The matrix coefficients for a 3x3 color matrix converting the image sensor color space into sRGB. The values are stored in row-scan order.
			colorMatrix => ...
			`[`
			`  1.91431,`
			`  -0.576416,`
			`  -0.234131,`
			`  -0.30542,`
			`  1.38916,`
			`  -0.0966797,`
			`  0.126953,`
			`  -0.952881,`
			`  1.64917`
			`]`

Name	Type	Flags	Description
`config`	`dictionar`	`r--`	Return a configuration dictionary of all saveable parameters

config => ...

```
{
  "recSegments": 1,
  "recMode": "normal",
  "ioThresholdIo1": 2.49929,
  "ioThresholdIo2": 2.49929,
  "ioMappingCombOr2": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingCombOr3": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingStopRec": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "exposureMode": "normal",
  "ioMappingToggleSet": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingCombOr1": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingCombXor": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingGate": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "recTrigDelay": 0,
  "currentGain": 2,
  "recPreBurst": 1,
  "ioMappingToggleFlip": {
    "source": "none",
    "debounce": false,
```

Name	Type	Flags	Description
`currentGain`	`float`	`rwn`	The current gain of the image sensor as a linear multiplier of `sensorIso`. currentGain => 2
`currentIso`	`float`	`rw-`	The ISO number of the image sensor at the current current gain. currentIso => 640
`dateTime`	`string`	`r--`	The current date and time in ISO-8601 format. dateTime => "2020-04-15T05:39:31.243589"
`digitalGain`	`float`	`rwn`	Digital image gain applied during video processing. digitalGain => 1
`disableRingBuffer`	`boolean`	`rw-`	When true, the camera will stop recording once the RAM buffer is full instead of looping over. By default, the camera will enable the ring buffer, so once the maximum record length has been reached, the camera will overwrite the oldest footage in the recording in normal recording mode, or overwrite the oldest segment in sedgmented recording mode. disableRingBuffer => false
`exposureMax`	`int`	`r-n`	The maximum possible time, in nanoseconds, that the image sensor is capable of exposing a frame for at the current `resolution` and `framePeriod`. exposureMax => 929900
`exposureMin`	`int`	`r-n`	The minimum possible time, in nanoseconds, that the image sensor is capable of exposing a frame for at the current `resolution` and `framePeriod`. exposureMin => 1000

Name	Type	Flags	Description
`exposureMode`	`string`	`rwn`	Mode in which frame timing and exposure should operate. • **normal**: Frame and exposure timing operate on fixed periods and are free-running. • **shutterGating**: Frame starts on the rising edge of the trigger signal, and exposes the frame for as long as the trigger signal is held high, regardless of the `exposurePeriod` property. Once readout completes, the camera will wait for another rising edge before starting the next frame. When in this mode, the `framePeriod` property constrains the minimum time between frames. • **frameTrigger**: Frame starts on the rising edge of the trigger signal, and exposes the frame for `exposurePeriod` nanoseconds. Once readout completes, the camera will wait for another rising edge before starting the next frame. In this mode, the `framePeriod` property constrains the minimum time between frames. exposureMode => "normal"
`exposureNormalized`	`float`	`rw-`	The current exposure time rescaled between `exposureMin` and `exposureMax`. This value is 0 when exposure is at minimum, and increases linearly until exposure is at maximum, when it is 1.0. exposureNormalized => 1
`exposurePercent`	`float`	`rw-`	The current exposure time rescaled between `exposureMin` and `exposureMax`. This value is 0% when exposure is at minimum, and increases linearly until exposure is at maximum, when it is 100%. exposurePercent => 100
`exposurePeriod`	`int`	`rwn`	Minimum period, in nanoseconds, that the image sensor is currently exposing frames for. exposurePeriod => 929900
`externalPower`	`boolean`	`r-n`	True when the AC adaptor is present, and False when on battery power. externalPower => true

Name	Type	Flags	Description

`externalStorage`	`dictionar`	`r--`	The currently attached external storage partitions and their status. The sizes of the reported storage devices are in units of kB.
			externalStorage => ...
			`{`
			`  "mmcblk1p1": {`
			`    "device": "/dev/mmcblk1p1",`
			`    "description": "MMC/SD Card Partiton 1",`
			`    "mount": "/media/mmcblk1p1",`
			`    "fstype": "vfat"`
			`  }`
			`}`

`fanOverride`	`float`	`rwn`	Fan speed in the range of 0=off to 1.0=full, or -1 for automatic fan control. fanOverride => -1
`focusPeakingColor`	`string`	`rwn`	The color to display when focus peaking detects a sharp edge. • **black**: • **red**: • **cyan**: • **blue**: • **yellow**: • **magenta**: • **white**: • **green**: focusPeakingColor => "magenta"
`focusPeakingLevel`	`float`	`rwn`	Edge sensitivity at which focus peaking is detected, with 0.0 disabling focus peaking and 1.0 for maximum sensitivity. focusPeakingLevel => 0
`framePeriod`	`int`	`rwn`	The time, in nanoseconds, to record a single frame. framePeriod => 935455
`frameRate`	`float`	`rw-`	The estimated estimated recording rate in frames per second (reciprocal of `framePeriod`). frameRate => 1069

Name	Type	Flags	Description
`ioDelayTime`	`float`	`rw-`	Delay time, in seconds, for the programmable delay block ioDelayTime => 0

Name	Type	Flags	Description
ioDetailedStatus	dictionar	r--	Detailed status of the IO block.

Values	Type	Description
detailedComb	dict	Dictionary of booleans showing the internal state of the combinatorial logic block.
edgeTimers	dict	Dictionary containing the time in clock cycles since the last rising and falling edges were measured for each output signal.
output	dict	Dictionary of booleans showing the state of all the output signals from the IO block.
sources	dict	The contents of the ioSouceStatus parameter.

ioDetailedStatus => ...

```
{
  "detailedComb": {
    "or1": false,
    "or2": false,
    "or3": false,
    "and": true,
    "xor": false
  },
  "edgeTimers": {
    "stop": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "interrupt": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "shutter": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "io1": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "io2": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "start": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "toggle": {
      "rising": 42.9497,
      "falling": 42.9497
```

Name	Type	Flags	Description
`ioMapping`	`dictionar`	`rw-`	Legacy interface to the IO block. This parameter contains a complex dictionary that both configures and describes the entire IO block in a single `set` operation. It is difficult to describe all of the nuances in which this parameter operates, so we recommend using the other IO block parameters to achieve your goal instead. ioMapping => ...

```
{
  "combAnd": {
    "source": "alwaysHigh",
    "debounce": false,
    "invert": false
  },
  "delay": {
    "delayTime": 0,
    "source": "comb",
    "debounce": false,
    "invert": false
  },
  "toggleSet": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "gate": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "toggleFlip": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "start": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "shutter": {
    "shutterTriggersFrame": false,
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "combXOr": {
    "source": "none",
    "debounce": false,
    "invert": false
  }
```

Name	Type	Flags	Description

ioMappingCombAnd	dictionar	rwn	Combinatorial block AND input configuration
			ioMappingCombAnd => ...
			{
			"source": "alwaysHigh",
			"debounce": false,
			"invert": false
			}

ioMappingCombOr1	dictionar	rwn	Combinatorial block OR input 1 configuration
			ioMappingCombOr1 => ...
			{
			"source": "none",
			"debounce": false,
			"invert": false
			}

ioMappingCombOr2	dictionar	rwn	Combinatorial block OR input 2 configuration
			ioMappingCombOr2 => ...
			{
			"source": "none",
			"debounce": false,
			"invert": false
			}

Name	Type	Flags	Description

`ioMappingCombOr3`	`dictionar`	`rwn`	Combinatorial block OR input 3 configuration
			ioMappingCombOr3 => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

`ioMappingCombXor`	`dictionar`	`rwn`	Combinatorial block XOR input configuration
			ioMappingCombXor => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

`ioMappingDelay`	`dictionar`	`rwn`	Programmable delay block input configuration
			ioMappingDelay => ...
			`{`
			`  "source": "comb",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

Name	Type	Flags	Description

ioMappingGate	dictionar	rwn	Gate input signal configuration
			ioMappingGate => ...
			{
			  "source": "none",
			  "debounce": false,
			  "invert": false
			}

ioMappingIo1	dictionar	rwn	Ouput driver 1 configuration
			ioMappingIo1 => ...
			{
			  "drive": 1,
			  "source": "alwaysHigh",
			  "debounce": false,
			  "invert": false
			}

ioMappingIo2	dictionar	rwn	Output driver 2 configuration
			ioMappingIo2 => ...
			{
			  "drive": 0,
			  "source": "alwaysHigh",
			  "debounce": false,
			  "invert": false
			}

Name	Type	Flags	Description

`ioMappingShutter`	`dictionar`	`rwn`	Timing block shutter control signal configuration
			ioMappingShutter => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

`ioMappingStartRec`	`dictionar`	`rwn`	Recording start signal configuration
			ioMappingStartRec => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

`ioMappingStopRec`	`dictionar`	`rwn`	Recording stop signal configuration
			ioMappingStopRec => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

Name	Type	Flags	Description

`ioMappingToggleClea`	`dictionar`	`rwn`	Toggle/flip-flop block CLEAR input configuration
			ioMappingToggleClear => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

`ioMappingToggleFlip`	`dictionar`	`rwn`	Toggle/flip-flop block FLIP input configuration
			ioMappingToggleFlip => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

`ioMappingToggleSet`	`dictionar`	`rwn`	Toggle/flip-flop block SET input configuration
			ioMappingToggleSet => ...
			`{`
			`  "source": "none",`
			`  "debounce": false,`
			`  "invert": false`
			`}`

Name	Type	Flags	Description

`ioMappingTrigger`	`dictionar`	`rwn`	Recording trigger signal configuration
			ioMappingTrigger => ...
			`{ "source": "io1", "debounce": true, "invert": true }`

`ioOutputStatus`	`dictionar`	`r--`	The output signals from the IO block and their current values.
			ioOutputStatus => ...
			`{ "gate": false, "delay": false, "start": false, "comb": false, "shutter": false, "toggle": true, "stop": false, "io1": true, "io2": true }`

<table>
<thead>
<tr><th>Name</th><th>Type</th><th>Flags</th><th>Description</th></tr>
</thead>
<tbody>
<tr><td>ioSourceStatus</td><td>dictionar</td><td>r--</td><td>The available IO signals and their current values.
ioSourceStatus => ...
<pre>
{
  "io3": false,
  "nextSeg": false,
  "delay": false,
  "io1": true,
  "dispFrame": false,
  "alwaysHigh": true,
  "none": false,
  "comb": false,
  "shutter": true,
  "toggle": true,
  "endRec": false,
  "timingIo": true,
  "recording": false,
  "software": false,
  "startRec": false,
  "io2": false
}
</pre></td></tr>
<tr><td>ioStatusSourceIo1</td><td>boolean</td><td>r--</td><td>The current logic level seen on the IO input 1 (BNC jack).
ioStatusSourceIo1 => true</td></tr>
<tr><td>ioStatusSourceIo2</td><td>boolean</td><td>r--</td><td>The current logic level seen on IO input 2 (green IO connector).
ioStatusSourceIo2 => false</td></tr>
<tr><td>ioStatusSourceIo3</td><td>boolean</td><td>r--</td><td>The current logic levle seeon on IO input 3 (opto-isolated input).
ioStatusSourceIo3 => false</td></tr>
<tr><td>ioThresholdIo1</td><td>float</td><td>rwn</td><td>Voltage threshold at which trigger input signal 1 should go high.
ioThresholdIo1 => 2.49929</td></tr>
<tr><td>ioThresholdIo2</td><td>float</td><td>rwn</td><td>Voltage threshold at which trigger input signal 2 should go high.
ioThresholdIo2 => 2.49929</td></tr>
</tbody>
</table>

Name	Type	Flags	Description
`lastShutdownReason`	`string`	`r--`	The reason for the last shutdown that happened. `lastShutdownReason => "97: PwrBtn, Software, PMIC Ack"`
`minFramePeriod`	`int`	`r-n`	The minimum frame period, in nanoseconds, at the current resolution settings. `minFramePeriod => 934922`

`miscScratchPad`	`dictionar`	`rwn`	A dictionary of arbitrary values that can be stored in the camera.
			`miscScratchPad => ...`
			`{`
			`  "empty": 1`
			`}`

`networkHostname`	`string`	`rw-`	Hostname to be used for dhcp requests and to be displayed on the command line. `networkHostname => "chronos"`
`overlayEnable`	`boolean`	`rwn`	Enabled the overlay text box when in playback mode `overlayEnable => false`
`overlayFormat`	`string`	`rwn`	Format string for the overlay text box `overlayFormat => "%.6h/%.6z Sg=%g/%i T=%.8Ss"`
`overlayPosition`	`string`	`rwn`	Location in the video stream to position the overlay textbox. This can take the values "top", "bottom" or a position of the form HPOSxVPOS. `overlayPosition => "bottom"`
`playbackLength`	`int`	`rwn`	The number of frames which should be replayed when in playback mode. `playbackLength => 0`
`playbackPosition`	`int`	`rw-`	The current frame being display when the camera is in playback mode. `playbackPosition => 0`

Name	Type	Flags	Description
`playbackRate`	`int`	`rwn`	The rate at which video is being replayed when in playback mode. playbackRate => 0
`playbackStart`	`int`	`rwn`	The starting frame from which video should be replayed when in playback mode. playbackStart => 0
`pmicFirmwareVersio`	`string`	`r--`	The Power Management IC's firmware version. pmicFirmwareVersion => "9"
`powerOffWhenMainsL`	`boolean`	`rwn`	True if the camera should power itself down when disconnected from mains power. powerOffWhenMainsLost => false
`powerOnWhenMainsCo`	`boolean`	`rwn`	True if the camera should power itself on when plugged into mains power. powerOnWhenMainsConnected => false
`recMaxFrames`	`int`	`rwn`	Limit on the maximum number of frames for the recording sequencer to use. recMaxFrames => 17470
`recMode`	`string`	`rwn`	Mode in which the recording sequencer stores frames into video memory. • **normal**: Frames are saved continuously into a ring buffer of up to `recMaxFrames` in length until the recording is terminated by the recording end trigger. • **burst**: Each rising edge of the recording trigger starts a new segment in video memory, with frames being saved for as long as the recording trigger is active. • **segmented**: Up to `recMaxFrames` of video memory is divided into `recSegments` number of of ring buffers. The camera saves video into one ring buffer at a time, switching to the next ring buffer at each recording trigger. recMode => "normal"
`recPreBurst`	`int`	`rwn`	The number of frames leading up to the trigger rising edge to save when in 'burst' recording mode. recPreBurst => 1
`recSegments`	`int`	`rwn`	The number of segments used by the recording sequencer when in 'segmented' recording mode. recSegments => 1

Name	Type	Flags	Description
recTrigDelay	int	rwn	The number of frames to delay the trigger rising edge by in 'normal' and 'segmented' recording modes. recTrigDelay => 0
resolution	dictiona	rwn	Resolution geometry at which the image sensor should capture frames. The optional hOffset and vOffset parameters allow the user to select where on the sensor to position the frame when operating at a cropped resolution. If not provided when setting, the camera will attempt to centre the cropped image on the image sensor. When setting resolution, the minFrameTime may be optionally provided to allow the image sensor to better tune itself for the desired frame period. When omitted, it is assumed that the sensor will tune itself for its maximum framerate.

Values	Type	Description
hOffset	int, optional	Horizontal offset, in pixels, from the top left of the full frame at which the first pixel will be read out.
hRes	int	Horizontal resolution of the catpured image, in pixels.
minFrameTime	float, optional	The minimum frame time, in seconds, that the image sensor is capable of recording frames when at this resolution configuration.
vDark	int, optional	The number of vertical dark rows to read out.
vOffset	int, optional	Vertical offset, in pixels, from the top left of the full frame at which the first pixel will be read out.
vRes	int	Vertical resolution of the captured image, in pixels.

resolution => ...

```
{
  "vRes": 1024,
  "minFrameTime": 0.000934922,
  "vOffset": 0,
  "hRes": 1280,
  "hOffset": 0,
  "vDarkRows": 0,
  "bitDepth": 12
}
```

Name	Type	Flags	Description
sensorBitDepth	int	r--	Number of bits per pixel sampled by the image sensor. sensorBitDepth => 12

Name	Type	Flags	Description
`sensorColorPattern`	`string`	`r--`	String describing the color filter array pattern of the image sensor. For example, a typical 2x2 Bayer pattern sensor might have a value of 'GRBG', while a monochrome image sensor would have a value of 'mono'. sensorColorPattern => "GRBG"
`sensorHIncrement`	`int`	`r--`	Minimum step size allowed, in pixels, for changes in the horizontal resolution of the image sensor. sensorHIncrement => 16
`sensorHMax`	`int`	`r--`	Maximum horizontal resolution, in pixels, of the active area of the image sensor. sensorHMax => 1280
`sensorHMin`	`int`	`r--`	Minimum horizontal resolution, in pixels, of the active area of the image sensor. sensorHMin => 192
`sensorIso`	`int`	`r--`	ISO number of the image sensor with nominal (0dB) gain applied. sensorIso => 320
`sensorMaxGain`	`int`	`r--`	Maximum gain of the image sensor as a linear muliplier of the `sensorISO`. sensorMaxGain => 16
`sensorName`	`string`	`r--`	Descriptive name of the image sensor. sensorName => "LUX1310"
`sensorPixelRate`	`float`	`r--`	Approximate throughput of the image sensor in pixels per second. sensorPixelRate => 1401980000
`sensorTemperature`	`float`	`r--`	The temperature, in degrees Celcius, measured near the image sensor. sensorTemperature => 38.9961
`sensorVDark`	`int`	`r--`	Maximum vertical resolution, in pixels, of the optical black regions of the sensor. sensorVDark => 8

<table>
<tr><th>Name</th><th>Type</th><th>Flags</th><th>Description</th></tr>
<tr><td>sensorVIncrement</td><td>int</td><td>r--</td><td>Minimum step size allowed, in pixels, for changes in the vertical resolution of the image sensor.
sensorVIncrement => 2</td></tr>
<tr><td>sensorVMax</td><td>int</td><td>r--</td><td>Maximum vertical resolution, in pixels, of the active area of the image sensor.
sensorVMax => 1024</td></tr>
<tr><td>sensorVMin</td><td>int</td><td>r--</td><td>Minimum vertical resolution, in pixels, of the active area of the image sensor.
sensorVMin => 32</td></tr>
<tr><td>shippingMode</td><td>boolean</td><td>rwn</td><td>True when the camera is configured for shipping mode
shippingMode => false</td></tr>
<tr><td>shutterAngle</td><td>float</td><td>rw-</td><td>The angle in degrees for which frames are being exposed relative to the frame time.
shutterAngle => 357.862</td></tr>
<tr><td>state</td><td>string</td><td>r-n</td><td>The current operating state of the camera.
<table>
<tr><th>Values</th><th>Type</th><th>Description</th></tr>
<tr><td>analogCal</td><td>undefined</td><td>The camera is currently performing analog calibration of the image sensor.</td></tr>
<tr><td>blackCal</td><td>undefined</td><td>The camera is currently calibrating using a dark reference image.</td></tr>
<tr><td>idle</td><td>undefined</td><td>The camera is powered up and operating, but not doing anything.</td></tr>
<tr><td>recording</td><td>undefined</td><td>The camera is running a recording program to save images into video memory.</td></tr>
<tr><td>reset</td><td>undefined</td><td>The camera is in the process of resetting the FPGA and image sensor.</td></tr>
</table>
state => "idle"</td></tr>
<tr><td>systemTemperature</td><td>float</td><td>r--</td><td>The temperature, in degrees Celcius, measured near the main processor.
systemTemperature => 47</td></tr>
<tr><td>totalFrames</td><td>int</td><td>r--</td><td>Total number of frames of recorded video that are have been saved into memory.
totalFrames => 872</td></tr>
</table>

Name	Type	Flags	Description
`totalSegments`	`int`	`r--`	Total number of video segments that have been saved into memory. totalSegments => 1

`videoConfig`	`dictionar`	`r--`	Dictionary of parameters saved persistently by the video system.
			videoConfig => ...
			`{`
			`  "overlayFormat": "%.6h/%.6z Sg=%g/%i T=%.8Ss",`
			`  "overlayEnable": false,`
			`  "overlayPosition": "bottom",`
			`  "focusPeakingLevel": 0,`
			`  "zebraLevel": 0,`
			`  "focusPeakingColor": "magenta"`
			`}`

`videoSegments`	`dictionar`	`r--`	Array of video segments, describing the size and metadata of that has been recorded.
			videoSegments => ...
			`[`
			`  {`
			`    "interval": 0.000935522,`
			`    "offset": 0,`
			`    "exposure": 0.000929933,`
			`    "length": 872`
			`  }`
			`]`

`videoState`	`string`	`r-n`	Current state of the video system. • **live**: • **filesave**: • **play**: • **paused**: videoState => "live"
`videoZoom`	`float`	`rwn`	Video scaling ratio to apply to the video stream (1.0 = fit to screen) videoZoom => 1

Name	Type	Flags	Description

`wbColor`	`array[fl`	`rwn`	The Red, Green and Blue gain coefficients to achieve white balance.
			wbColor => ...
			`[`
			`  1.52979,`
			`  1,`
			`  1.34985`
			`]`

`wbCustomColor`	`array[fl`	`rwn`	The Red, Green and Blue gain coefficients last computed by `startWhiteBalance()`.
			wbCustomColor => ...
			`[`
			`  1,`
			`  1,`
			`  1`
			`]`

`wbTemperature`	`int`	`rwn`	Color temperature, in degrees Kelvin, to use for white balance. wbTemperature => 8000
`zebraLevel`	`float`	`rwn`	Pixel threshold at which zebra striping is enabled. Values close to 0.0 only trigger zebra stripes near saturation, and values near 1.0 would enable zebra stripes even when the image is black. zebraLevel => 0