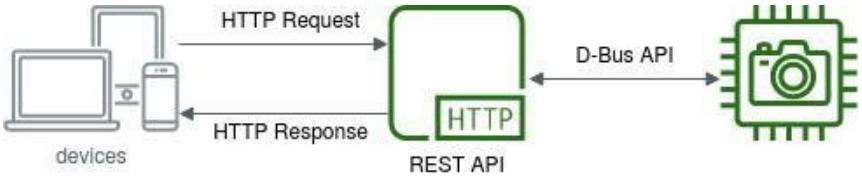
# Overview

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 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.

**Note:** Use the IP Address specified in the Util -> Network screen when accessing the camera via Ethernet

# 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.

**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** | 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 methods 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 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 what 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 recorded 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 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 retrieve 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 maximum exposure period in nanoseconds, that the image sensor can expose a frame for if the framePeriod was set equal to minFramePeriod. |
| **exposureMin** | int | The minimum exposure period in nanoseconds that the image sensor can expose 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 trigger 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 themselves (default: true). |
| **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 parameter 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 begins 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 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 (e.g.: 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 file save is completed, the video system will exit the `filesave` state and revert 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).<br>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 begins<br>(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 hot plug 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 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 programs the recording sequencer and starts 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 begins 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 OK.

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 | Vertical 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 file save operation

When the video system has started a file save operation, it can take a very long time to complete depending on the quantity of footage being saved, and the speed of media to which it is being written.

If an 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 file save 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 terminates 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.

**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 is completed successfully then the data will contain a dictionary with the name of the method completed, and the new state of the camera.

If the procedure is completed with an error, then the dictionary will also contain an `error` with the type of error that occurred, and optionally a `message` with a human-readable description of the error.

**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 occurred during the asynchronous call (optional) |

| | |
|---|---|
| **message** | A human-readable string describing the cause of the error |

# Parameters

The Chronos API exposes 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 **`get`** is true. |
| **PUT** | `/control/p/{name}` | Set the value of a single parameter by name if the **`set`** flag is true. |
| **POST** | `/control/p` | Update a collection of parameters together |

## backlightEnabled

**Type:** Boolean

**Get:** true
**Set:** true
**Notifies:** true

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.

```javascript
true
```

## batteryChargeNormalized

**Type:** Double

**Get:** true
**Set:** false
**Notifies:** false

Estimated battery charge, with 0.0 being depleted and 1.0 being fully charged.

```javascript
1
```

## batteryChargePercent

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Estimated battery charge, with 0% being depleted and 100% being fully charged.

```
JavaScript
100
```

## batteryCritical

**Type:** Boolean

**Get:** true
**Set:** false
**Notifies:** true

True when the battery voltage is critically low and a powerdown is imminent

```
JavaScript
false
```

## batteryPresent

**Type:** Boolean

**Get:** true
**Set:** false
**Notifies:** true

True when the battery is installed, and False when the camera is only running on adaptor power

```
JavaScript
true
```

## batteryVoltage

**Type:** Double

**Get:** true
**Set:** false
**Notifies:** false

The voltage that is currently being output by the battery. A fully charged battery outputs between 12V and 12.5V.

```javascript
12.225
```

## calSuggested

**Type:** Boolean

**Get:** true
**Set:** false
**Notifies:** true

True when the calibration of the camera needs updating.

```javascript
false
```

## cameraApiVersion

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

Version string of the chronos module

```javascript
"0.7.0"
```

## cameraDescription

**Type:** String

**Get:** true
**Set:** true
**Notifies:** true

Descriptive string assigned by the user

```JavaScript
"Chronos SN:00000"
```

## cameraFpgaVersion

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

Version string of the FPGA bitstream that is currently running

```JavaScript
"3.24"
```

## cameraIdNumber

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

Unique camera number assigned by the user

```JavaScript
0
```

## cameraMaxFrames

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** true

The maximum number of frames the camera's memory can save at the current resolution.

```JavaScript
11038
```

## cameraMemoryGB

**Type:** Double

**Get:** true
**Set:** false
**Notifies:** false

Amount of video memory attached to the FPGA in GiB

```JavaScript
32
```

## cameraModel

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

Camera model name

```JavaScript
"CR21-1.0"
```

## cameraSerial

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

Unique camera serial number

```JavaScript
"00000"
```

## cameraTallyMode

**Type:** String

**Get:** true
**Set:** true
**Notifies:** true

Mode in which the recording LEDs should operate.

```javascript
"auto"
```

## colorMatrix

**Type:** Array of Doubles

**Get:** true
**Set:** true
**Notifies:** true

The matrix coefficients for a 3x3 color matrix converting the image sensor color space into sRGB. The values are stored in row-scan order.

```javascript
[
  2.20679,
  -0.863281,
  -0.239258,
  -0.375488,
  1.63477,
  -0.271973,
  0.0136719,
  -0.911377,
  1.72046
]
```

## config

**Type:** Dictionary

**Get:** true
**Set:** false
**Notifies:** false

Return a configuration dictionary of all saveable parameters

JavaScript


```
{
  "recSegments": 5,
  "recMode": "normal",
  "ioMappingToggleSet": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioThresholdIo2": 2.50271,
  "ioMappingCombOr2": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingCombOr3": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingStopRec": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "recTrigDelay": 0,
  "ioMappingGate": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingCombOr1": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "exposureMode": "normal",
  "ioMappingCombXor": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioThresholdIo1": 2.50271,
  "currentGain": 1.07143,
```

```
  "recPreBurst": 1,
  "ioMappingToggleFlip": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioDelayTime": 0,
  "cameraDescription": "Chronos SN:00000",
  "wbCustomColor": [
    1.25822,
    1,
    1.10377
  ],
  "miscScratchPad": {
    "empty": 1
  },
  "ioMappingStartRec": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingIo2": {
    "source": "alwaysHigh",
    "drive": 0,
    "debounce": true,
    "invert": true
  },
  "ioMappingIo1": {
    "source": "alwaysHigh",
    "drive": 0,
    "debounce": true,
    "invert": true
  },
  "colorMatrix": [
    2.20679,
    -0.863281,
    -0.239258,
    -0.375488,
    1.63477,
    -0.271973,
    0.0136719,
    -0.911377,
    1.72046
  ],
```

```
  "ioMappingTrigger": {
    "source": "io1",
    "debounce": true,
    "invert": true
  },
  "disableRingBuffer": 0,
  "ioMappingDelay": {
    "source": "comb",
    "debounce": false,
    "invert": false
  },
  "resolution": {
    "bitDepth": 12,
    "vOffset": 0,
    "vRes": 1080,
    "minFrameTime": 0.000999893,
    "hOffset": 0,
    "hRes": 1920,
    "vDarkRows": 0
  },
  "recMaxFrames": 11038,
  "wbTemperature": 0,
  "ioMappingToggleClear": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "ioMappingCombAnd": {
    "source": "alwaysHigh",
    "debounce": false,
    "invert": false
  },
  "cameraIdNumber": 0,
  "cameraTallyMode": "auto",
  "exposurePeriod": 500950,
  "digitalGain": 1,
  "ioMappingShutter": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "framePeriod": 999893
}
```

## currentGain

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

The current gain of the image sensor as a linear multiplier of `sensorIso`.

```
JavaScript
1.07143
```

## currentIso

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** false

The ISO number of the image sensor at the current current gain.

```
JavaScript
621.429
```

## dateTime

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

The current date and time in ISO-8601 format.

```
JavaScript
"2023-09-25T10:42:43.294041"
```

## digitalGain

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

Digital image gain applied during video processing.

```
JavaScript
1
```

## disableRingBuffer

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** false

When true, the camera will stop recording once the RAM buffer is full instead of looping over.

```
JavaScript
0
```

## exposureMax

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** true

The maximum possible time, in nanoseconds, that the image sensor is capable of exposing

```
JavaScript
993226
```

## exposureMin

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** true

The minimum possible time, in nanoseconds, that the image sensor is capable of exposing

```JavaScript
1000
```

## exposureMode

**Type:** String

**Get:** true
**Set:** true
**Notifies:** true

Mode in which frame timing and exposure should operate.

```JavaScript
"normal"
```

## exposureNormalized

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** false

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.

```JavaScript
0.503877
```

## exposurePercent

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** false

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%.

```JavaScript
50.3877
```

## exposurePeriod

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

Minimum period, in nanoseconds, that the image sensor is currently exposing frames for.

```JavaScript
500950
```

## externalPower

**Type:** Boolean

**Get:** true
**Set:** false
**Notifies:** true

True when the AC adaptor is present, and False when on battery power.

```JavaScript
true
```

## externalStorage

**Type:** Dictionary

**Get:** true
**Set:** false
**Notifies:** false

The currently attached external storage partitions and their status. The sizes of the reported storage devices are in units of kB.

JavaScript

```
{
  "mmcblk1p1": {
    "device": "/dev/mmcblk1p1",
    "description": "MMC/SD Card Partiton 1",
    "mount": "/media/mmcblk1p1",
    "fstype": "vfat"
  },
  "sda1": {
    "device": "/dev/sda1",
    "description": "ATA TEAM T253X2001T Partiton 1",
    "mount": "/media/sda1",
    "fstype": "vfat"
  }
}
```

## fanOverride

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

Fan speed in the range of 0=off to 1.0=full, or -1 for automatic fan control.

JavaScript

```
-1
```

## focusPeakingColor

**Type:** String

**Get:** true
**Set:** true
**Notifies:** true

The color to display when focus peaking detects a sharp edge.

JavaScript

```
"cyan"
```

## focusPeakingLevel

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

Edge sensitivity at which focus peaking is detected, with 0.0 disabling focus peaking and 1.0 for maximum sensitivity.

```javascript
0
```

## framePeriod

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

The time, in nanoseconds, to record a single frame.

```javascript
999893
```

## frameRate

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** false

The estimated recording rate in frames per second (reciprocal of `framePeriod`).

```javascript
1000.11
```

## ioDelayTime

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

Delay time, in seconds, for the programmable delay block

```
JavaScript
0
```

# ioDetailedStatus

**Type:** Dictionary

**Get:** true
**Set:** false
**Notifies:** false

Detailed status of the IO block.

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

```
JavaScript
{
  "outputs": {
    "gate": false,
    "delay": false,
    "start": false,
    "comb": false,
    "shutter": false,
    "toggle": false,
    "stop": false,
    "io1": false,
    "io2": false
  },
  "edgeTimers": {
    "shutter": {
      "rising": 42.9497,
```

```
      "falling": 42.9497
    },
    "interrupt": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "stop": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "io1": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "toggle": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "start": {
      "rising": 42.9497,
      "falling": 42.9497
    },
    "io2": {
      "rising": 42.9497,
      "falling": 42.9497
    }
  },
  "detailedComb": {
    "or1": false,
    "or2": false,
    "or3": false,
    "xor": false,
    "and": true
  },
  "sources": {
    "io3": false,
    "nextSeg": false,
    "delay": false,
    "timingIo": true,
    "dispFrame": false,
    "alwaysHigh": true,
    "comb": false,
    "none": false,
    "shutter": true,
```

```
    "toggle": false,
    "endRec": false,
    "io1": false,
    "recording": false,
    "software": false,
    "startRec": false,
    "io2": false
  }
}
```

## ioMapping

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** false

Legacy interface to the IO block.

JavaScript

```
{
  "combAnd": {
    "source": "alwaysHigh",
    "debounce": false,
    "invert": false
  },
  "toggleSet": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "delay": {
    "source": "comb",
    "delayTime": 0,
    "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
  },
  "combOr2": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "combOr3": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "combOr1": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "toggleClear": {
    "source": "none",
    "debounce": false,
    "invert": false
  },
  "stop": {
    "source": "none",
    "debounce": false,
```

```
    "invert": false
  },
  "trigger": {
    "source": "io1",
    "debounce": true,
    "invert": true
  },
  "io1In": {
    "threshold": 2.50271
  },
  "io2In": {
    "threshold": 2.50271
  },
  "io1": {
    "source": "alwaysHigh",
    "driveStrength": 0,
    "debounce": true,
    "invert": true
  },
  "io2": {
    "source": "alwaysHigh",
    "driveStrength": 0,
    "debounce": true,
    "invert": true
  }
}
```

## ioMappingCombAnd

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Combinatorial block AND input configuration

JavaScript

```
{
  "source": "alwaysHigh",
  "debounce": false,
  "invert": false
}
```

# ioMappingCombOr1

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Combinatorial block OR input 1 configuration

```JavaScript
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

# ioMappingCombOr2

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Combinatorial block OR input 2 configuration

```JavaScript
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

# ioMappingCombOr3

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Combinatorial block OR input 3 configuration

JavaScript

```
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingCombXor

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Combinatorial block XOR input configuration

JavaScript

```
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingDelay

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Programmable delay block input configuration

JavaScript

```
{
  "source": "comb",
  "debounce": false,
  "invert": false
}
```

## ioMappingGate

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Gate input signal configuration

```JavaScript
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingIo1

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Output driver 1 configuration

```JavaScript
{
  "source": "alwaysHigh",
  "drive": 0,
  "debounce": true,
  "invert": true
}
```

## ioMappingIo2

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Output driver 2 configuration

JavaScript

```
{
  "source": "alwaysHigh",
  "drive": 0,
  "debounce": true,
  "invert": true
}
```

## ioMappingShutter

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Timing block shutter control signal configuration

JavaScript

```
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingStartRec

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Recording start signal configuration

JavaScript

```
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingStopRec

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Recording stop signal configuration

```JavaScript
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingToggleClear

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Toggle/flip-flop block CLEAR input configuration

```JavaScript
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingToggleFlip

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Toggle/flip-flop block FLIP input configuration

```JavaScript
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingToggleSet

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Toggle/flip-flop block SET input configuration

```JavaScript
{
  "source": "none",
  "debounce": false,
  "invert": false
}
```

## ioMappingTrigger

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Recording trigger signal configuration

```JavaScript
{
  "source": "io1",
  "debounce": true,
  "invert": true
}
```

## ioOutputStatus

**Type:** Dictionary

**Get:** true
**Set:** false
**Notifies:** false

The output signals from the IO block and their current values.

```JavaScript
{
  "gate": false,
  "delay": false,
  "start": false,
  "comb": false,
  "shutter": false,
  "toggle": false,
  "stop": false,
  "io1": false,
  "io2": false
}
```

## ioSourceStatus

**Type:** Dictionary

**Get:** true
**Set:** false
**Notifies:** false

The available IO signals and their current values.

```JavaScript
{
  "io3": false,
  "nextSeg": false,
  "delay": false,
  "timingIo": false,
  "dispFrame": false,
  "alwaysHigh": true,
  "comb": false,
  "none": false,
  "shutter": true,
  "toggle": false,
  "endRec": false,
  "io1": false,
```

```
  "recording": false,
  "software": false,
  "startRec": false,
  "io2": false
}
```

## ioStatusSourceIo1

**Type:** Boolean

**Get:** true
**Set:** false
**Notifies:** false

The current logic level seen on the IO input 1 (BNC jack).

```JavaScript
false
```

## ioStatusSourceIo2

**Type:** Boolean

**Get:** true
**Set:** false
**Notifies:** false

The current logic level seen on IO input 2 (green IO connector).

```JavaScript
false
```

## ioStatusSourceIo3

**Type:** Boolean

**Get:** true
**Set:** false
**Notifies:** false

The current logic level seen on IO input 3 (opto-isolated input).

```JavaScript
false
```

## ioThresholdIo1

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

Voltage threshold at which trigger input signal 1 should go high.

```JavaScript
2.50271
```

## ioThresholdIo2

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

Voltage threshold at which trigger input signal 2 should go high.

```JavaScript
2.50271
```

## lastShutdownReason

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

The reason for the last shutdown that happened.

```JavaScript
"97: PwrBtn, Software, PMIC Ack"
```

## minFramePeriod

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** true

The minimum frame period, in nanoseconds, at the current resolution settings.

```javascript
999893
```

## miscScratchPad

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

A dictionary of arbitrary values that can be stored in the camera.

```javascript
{
  "empty": 1
}
```

## networkHostname

**Type:** String

**Get:** true
**Set:** true
**Notifies:** false

Hostname to be used for dhcp requests and to be displayed on the command line.

```javascript
chronos
```

## overlayEnable

**Type:** Boolean

**Get:** true
**Set:** true
**Notifies:** true

Enabled the overlay text box when in playback mode

```javascript
false
```

## overlayFormat

**Type:** String

**Get:** true
**Set:** true
**Notifies:** true

Format string for the overlay text box

```javascript
"%.6h/%.6z Sg=%g/%i T=%.8Ss"
```

## overlayPosition

**Type:** String

**Get:** true
**Set:** true
**Notifies:** true

Location in the video stream to position the overlay textbox. This can take the values "top", "bottom" or a position of the form HPOSxVPOS.

```javascript
"bottom"
```

## playbackLength

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

The number of frames which should be replayed when in playback mode.

```
JavaScript
11032
```

## playbackPosition

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** false

The current frame being displayed when the camera is in playback mode.

```
JavaScript
0
```

## playbackRate

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

The rate at which video is being replayed when in playback mode.

```
JavaScript
0
```

## playbackStart

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

The starting frame from which video should be replayed when in playback mode.

```JavaScript
0
```

## pmicFirmwareVersion

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

The Power Management Integrated Circuit's firmware version.

```JavaScript
"11"
```

## powerOffWhenMainsLost

**Type:** Boolean

**Get:** true
**Set:** true
**Notifies:** true

True if the camera should power itself down when disconnected from mains power.

```JavaScript
false
```

## powerOnWhenMainsConnected

**Type:** Boolean

**Get:** true
**Set:** true
**Notifies:** true

True if the camera should power itself on when plugged into mains power.

```javascript
false
```

## recMaxFrames

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

Limit on the maximum number of frames for the recording sequencer to use.

```javascript
11038
```

## recMode

**Type:** String

**Get:** true
**Set:** true
**Notifies:** true

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 ring buffers. The camera saves video into one ring buffer at a time, switching to the next ring buffer at each recording trigger.

```javascript
"normal"
```

## recPreBurst

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

The number of frames leading up to the trigger rising edge to save when in 'burst' recording mode.

```
JavaScript
1
```

## recSegments

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

The number of segments used by the recording sequencer when in 'segmented' recording mode.

```
JavaScript
5
```

## recTrigDelay

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

The number of frames to delay the trigger rising edge in 'normal' and 'segmented' recording modes.

```
JavaScript
0
```

## resolution

**Type:** Dictionary

**Get:** true
**Set:** true
**Notifies:** true

Resolution geometry at which the image sensor should capture frames.

```JavaScript
{
  "bitDepth": 12,
  "vOffset": 0,
  "vRes": 1080,
  "minFrameTime": 0.000999893,
  "hOffset": 0,
  "hRes": 1920,
  "vDarkRows": 0
}
```

## sensorBitDepth

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Number of bits per pixel sampled by the image sensor.

```JavaScript
12
```

## sensorColorPattern

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

String describing the color filter array pattern of the image sensor.

```JavaScript
"GRBG"
```

## sensorHIncrement

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Minimum step size allowed, in pixels, for changes in the horizontal resolution of the image sensor.

```
JavaScript
32
```

## sensorHMax

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Maximum horizontal resolution, in pixels, of the active area of the image sensor.

```
JavaScript
1920
```

## sensorHMin

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Minimum horizontal resolution, in pixels, of the active area of the image sensor.

```
JavaScript
640
```

## sensorIso

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

ISO number of the image sensor with nominal (0dB) gain applied.

```JavaScript
580
```

## sensorMaxGain

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Maximum gain of the image sensor as a linear multiplier of the `sensorISO`.

```JavaScript
16
```

## sensorName

**Type:** String

**Get:** true
**Set:** false
**Notifies:** false

Descriptive name of the image sensor.

```JavaScript
"LUX2100"
```

## sensorPixelRate

**Type:** Double

**Get:** true
**Set:** false
**Notifies:** false

Approximate throughput of the image sensor in pixels per second.

```JavaScript
2073920000
```

## sensorTemperature

**Type:** Double

**Get:** true
**Set:** false
**Notifies:** false

The temperature, in degrees Celsius, measured near the image sensor.

```JavaScript
45
```

## sensorVDark

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Maximum vertical resolution, in pixels, of the optical black regions of the sensor.

```JavaScript
8
```

## sensorVIncrement

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Minimum step size allowed, in pixels, for changes in the vertical resolution of the image sensor.

```JavaScript
2
```

## sensorVMax

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Maximum vertical resolution, in pixels, of the active area of the image sensor.

```
JavaScript
1080
```

## sensorVMin

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Minimum vertical resolution, in pixels, of the active area of the image sensor.

```
JavaScript
96
```

## shippingMode

**Type:** Boolean

**Get:** true
**Set:** true
**Notifies:** true

True when the camera is configured for shipping mode

```
JavaScript
false
```

## shutterAngle

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** false

The angle in degrees for which frames are being exposed relative to the frame time.

```javascript
180.365
```

## state

**Type:** String

**Get:** true
**Set:** false
**Notifies:** true

The current operating state of the camera.

- **analogCal**: The camera is currently performing analog calibration of the image sensor.
- **blackCal**: The camera is currently calibrating using a dark reference image.
- **idle**: The camera is powered up and operating, but not doing anything.
- **recording**: The camera is running a recording program to save images into video memory.
- **reset**: The camera is in the process of resetting the FPGA and image sensor.

```javascript
"idle"
```

## systemTemperature

**Type:** Double

**Get:** true
**Set:** false
**Notifies:** false

The temperature, in degrees Celsius, measured near the main processor.

```JavaScript
47.3
```

## totalFrames

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Total number of frames of recorded video that have been saved into memory.

```JavaScript
11032
```

## totalSegments

**Type:** Integer

**Get:** true
**Set:** false
**Notifies:** false

Total number of video segments that have been saved into memory.

```JavaScript
0
```

## videoConfig

**Type:** Dictionary

**Get:** true
**Set:** false
**Notifies:** false

Dictionary of parameters saved persistently by the video system.

JavaScript

```
{
  "overlayFormat": "%.6h/%.6z Sg=%g/%i T=%.8Ss",
  "overlayEnable": false,
  "overlayPosition": "bottom",
  "focusPeakingLevel": 0,
  "zebraLevel": 0,
  "focusPeakingColor": "cyan"
}
```

## videoSegments

**Type:** Dictionary

**Get:** true
**Set:** false
**Notifies:** false

Array of video segments, describing the size and metadata that has been recorded.

JavaScript

```
[]
```

## videoState

**Type:** String

**Get:** true
**Set:** false
**Notifies:** true

Current state of the video system.

- **live**
- **filesave**
- **play**
- **paused**

JavaScript

```
"live"
```

## videoZoom

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

Video scaling ratio to apply to the video stream (1.0 = fit to screen)

```javascript
1
```

## wbColor

**Type:** Array of Doubles

**Get:** true
**Set:** true
**Notifies:** true

The Red, Green and Blue gain coefficients to achieve white balance.

```javascript
[
  1.25806,
  1,
  1.10376
]
```

## wbCustomColor

**Type:** Array of Doubles

**Get:** true
**Set:** true
**Notifies:** true

The Red, Green and Blue gain coefficients last computed by `startWhiteBalance()`.

```javascript
[
  1.25822,
```

```
  1,
  1.10377
]
```

## wbTemperature

**Type:** Integer

**Get:** true
**Set:** true
**Notifies:** true

Color temperature, in degrees Kelvin, to use for white balance.

```JavaScript
0
```

## zebraLevel

**Type:** Double

**Get:** true
**Set:** true
**Notifies:** true

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.

```JavaScript
0
```