MRTech IFF SDK technical manual

Copyright © 2023 MRTech SK, s.r.o.

The information provided in this documentation is believed to be accurate and reliable as of the date provided.

Revision History

Number	Date	Description	Name
1.0	February 2022	initial release	MS
1.1	March 2022	updated component descriptions	MS
1.2	May 2022	added chain control interface description	MS
1.3	January 2022	IFF SDK release v1.3	MS
1.4	August 2023	IFF SDK release v1.4	MS
1.5	September 2023	IFF SDK release v1.5	MS
1.6	October 2023	IFF SDK release v1.6	MS
1.7	December 2023	IFF SDK release v1.7	MS

Contents

1 Introduction

MRTech IFF SDK provides an environment for creating image processing applications targeted for high-performance machine vision systems.

IFF SDK takes its name from Image Flow Framework (IFF) which has been developed and used by MRTech company for its machine vision projects since 2016.

The intended and structural purposes of the IFF SDK are to acquire, process, deliver images in the way the user wants, as efficiently as possible. With IFF SDK as MRTech team believes the users can achieve maximum performance for the chosen configuration of the image processing system.

All rights to IFF SDK belong to MRTech SK.

1.1 Documentation and Support

The manual explains how to install MRTech IFF SDK to run it successfully.

If you have not already used IFF SDK and performed the initial setup steps, see the Quick start guide.

A detailed description of the library components, their parameters, as well as examples of how to use the SDK effectively are given in the following sections.

Note
MRTech is constantly developing IFF SDK, so the manual can be subject to change.

For more information, or if the user needs support in using IFF SDK, please contact us.

1.2 Contact MRTech

MRTech SK, s.r.o.

- Web: https://mr-technologies.com/
- Email: support@mr-technologies.com

2 About IFF SDK

2.1 System requirements

Supported hardware platforms:

- 64-bit Intel x86 (also known as x86_64 or AMD64)
- 64-bit ARM (also known as ARM64 or AArch64)
  - main target is NVIDIA Jetson family

Supported operating systems:

- Linux
- Windows
- macOS (preliminary support)

Supported hardware acceleration devices:

- GPU
  - NVIDIA GPUs, including embedded Jetson platform, using CUDA API
- video encoding
  - discrete NVIDIA GPUs using NVENC API
  - embedded NVIDIA Jetson platform using V4L API
- video decoding
  - discrete NVIDIA GPUs using NVDEC API

2.2 Basic features

- Textual description of pipeline configuration that allows user to create image processing workflows of any complexity.
- A wide range of processing modules (e.g. demosaicing, video encoding) working out-of-the-box.
- Ability to export and import images from the SDK pipeline to the customer application.
- Control of pipeline parameters at runtime.
- Easy integration with OpenCV, third-party processing libraries, custom processing modules.
- Hardware and software accelerated image processing on NVIDIA GPUs.

2.3 Advantages

- Production-ready, high-quality code, successfully used in many projects.
- High-performance image processing with low latency and low overhead.
- SDK architecture, that makes it easy to develop and customize the target application.
- Technical support, consulting, assistance from MRTech in implementation (when necessary).

2.4 Concepts

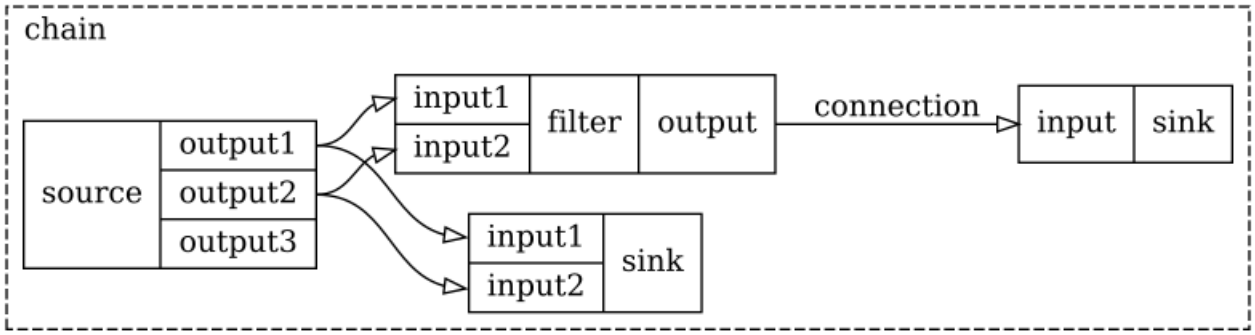

Figure 1: Pipeline

- IFF SDK purpose is to create and manage an image processing pipeline based on clear-text description in JSON format.
- Pipeline consists of one or more chains and images passing through them.
- Each chain is a directed acyclic graph defined by a list of elements and connections between their inputs and outputs.
- Element is an instantiation of specific IFF SDK component implementing some function (e.g. video encoding).
- Each component has a specific list of parameters, commands and callbacks.
- Element can have any number (zero or more) of inputs and outputs, defined by its type (component name) and configuration (parameters).
- Connection specifies that images from an output of one element will be passed to an input of another element.
- There must be exactly one connection per each existing input in a chain.
- Output on the other hand can be a source of any number (zero or more) of connections.
- Images are queued at inputs of elements and are dropped if queue exceeds the size specified in element parameters.
- Each image is defined by its metadata and a buffer (memory pointer) residing in some device (CPU or GPU RAM).
- All needed buffers are pre-allocated once pipeline parameters are determined, so out of memory situation is detected early.

3 Quick start

3.1 Dependencies

1. For CUDA edition: NVIDIA GPU drivers
2. For GenICam edition: camera vendor drivers and GenTL producer library, for example:
   a. pylon Camera Software Suite from Basler
3. For XIMEA edition: XIMEA software package

3.2 Package contents

1. `documentation` - this manual
2. `samples` - example source code from Sample applications
3. `sdk` - MRTech IFF SDK
   a. `include` - C header file
   b. `lib` - shared libraries
   c. `licenses_3rdparty` - license texts of third party software used by IFF SDK
4. `version.txt` - release number and edition information

3.3 Installation

1. Install packages listed in Dependencies.
2. Unpack the MRTech IFF SDK package.
3. Build a sample:
   - on Linux or in Windows developer shell:

```
cd samples/01_streaming
mkdir build
cd build
cmake ..
cmake --build .
```

   - in Microsoft Visual Studio: open `samples/01_streaming` folder and build as usual.
4. Edit configuration file `farsight.json` in `samples/01_streaming/build/bin` directory:
   - replace `<CHANGEME>` strings with correct values (IP address and camera serial number);
   - on Jetson change `NV12_BT709` to `YV12_BT709` as indicated by the inline comment;
   - adjust other settings, if you'd like.
5. Run the sample:
   - on Linux:

```
cd bin
./farsight
```

   - on Windows: execute `farsight.exe` in `samples/01_streaming/build/bin` directory.

4 IFF components

There are three kinds of IFF components: sources, sinks and filters.

Any kind of components shares two interfaces:

- `element` - this interface gives component an ability to be chained e.g. linked into processing chain
- `controllable` - an interface which gives an ability components parameters to be controlled in runtime

All components have following common parameters:

```
{
    "id": "comp_id",
    "type": "comp_type",
    "max_processing_count": 2
}
```

- `id`: ID of the component. Must be unique within given processing chain.
- `type`: type of the component (e.g. `xicamera`, `rtsp_stream`, `rtsp_source`, e.t.c.)
- `max_processing_count` (default 2): maximum number of frames that can be simultaneously processed by given instance of the component

---

4.1 Sources

Components of this kind inject data into the processing chain. They have no inputs, but only outputs. So this kind of component should be the initial element of the processing chain.

Common parameters for all sources are:

```
{
    "dispatch_control_mode": "subscription",
    "trigger_mode": {
        "mode": "free_run",
        "line": 0
    }
}
```

- `dispatch_control_mode` (default ”subscription”): start/stop dispatching mode, one of the following values:
  - `subscription`: automatically start dispatching when first consumer subscribed and stop when last consumer unsubscribed
  - `command`: explicitly start/stop dispatching by corresponding commands
- `trigger_mode`:
  - `mode` (default ”free_run”): trigger mode, one of the following values:
    - `free_run`: new frames are dispatched automatically
    - `software`: new frame is dispatched by `trigger` command

    * `hardware_rising`: new frame is dispatched when rising signal is detected on camera hardware trigger line
    * `hardware_falling`: new frame is dispatched when rising signal is detected on camera hardware trigger line
  - `line` (default 0): camera hardware trigger line number, only used for hardware trigger mode

Any source also supports the following two commands:

- `start`: makes source start dispatch images
- `stop`: makes source stop dispatch images
- `trigger`: makes source dispatch an image, if it's in software trigger mode

See `iff_execute()` for more details on command execution by elements.

4.1.1 `genicam`

GenICam camera.

JSON configuration:

```
{
    "id": "cam",
    "type": "genicam",
    "max_processing_count": 2,
    "dispatch_control_mode": "subscription",
    "cpu_device_id": "cpu_dev",
    "producer": "/opt/pylon/lib/gentlproducer/gtl/ProducerU3V.cti",
    "serial_number": "23096645",
    "max_open_retries": -1,
    "wait_after_error_sec": 3,
    "use_alloc": false,
    "min_buffers_queue_size": 1,
    "image_capture_timeout": 5000,
    "pixel_format": "BayerRG12p",
    "roi_region": {
        "offset_x": 0,
        "offset_y": 0,
        "width": 1920,
        "height": 1080
    },
    "custom_params": [
        { "DeviceLinkThroughputLimitMode": "On" },
        { "DeviceLinkThroughputLimit": 400000000 }
    ],
    "exposure": 10000,
    "gain": 0.0,
    "fps": 0.0,
    "auto_white_balance": false,
    "wb": {
        "r": 1.0,
        "g1": 1.0,
        "g2": 1.0,
```

```
        "b": 1.0,
        "r_off": 0,
        "g_off": 0,
        "b_off": 0
    }
}
```

parameters

- `cpu_device_id`: CPU device ID
- `producer`: path to GenTL producer library (usually comes with the camera vendor's software package)
- `serial_number`: serial number of GenICam camera
- `max_open_retries` (default -1): the maximum number of retries to open the camera before giving up (and transitioning to the `disconnected` state), negative value means unlimited
- `wait_after_error_sec` (default 3): time in seconds between attempts to open the camera
- `use_alloc` (default false): whether to allocate buffers using `DSAllocAndAnnounceBuffer` GenTL producer function
- `min_buffers_queue_size` (default `max_processing_count`-1): minimal number of available buffers in acquisition queue
- `image_capture_timeout` (default 5000): get image timeout in milliseconds
- `pixel_format`: camera output GenICam image pixel format
- `roi_region` (optional): camera ROI, not modified by default
- `custom_params` (optional): custom GenICam camera parameters
- `exposure` (default 10000): camera exposure in microseconds, can be modified at runtime
- `gain` (default 0.0): camera gain in dB, can be modified at runtime
- `fps` (default 0.0): camera FPS limit, zero means unlimited (free run), can be modified at runtime
- `auto_white_balance` (default false): enable GenICam camera auto white balance, can be modified at runtime
- `wb` (optional): default white balance settings, all parameters are optional (by default gains are set to 1.0 and offsets to 0), green coefficients can be set either together (`g` and `g_off`) or separately (`g1`, `g2`, `g1_off` and `g2_off`, which override `g` and `g_off`), can be modified at runtime

4.1.2 `raw_frame_player`

Reads all image files from specified directory and dispatches them to subscribers with given FPS.

JSON configuration:

```
{
    "id": "reader",
    "type": "raw_frame_player",
    "dispatch_control_mode": "subscription",
    "trigger_mode": {
        "mode": "free_run"
    }
    "cpu_device_id": "cpu_dev",
    "directory": "/path/to/frames/directory",
    "offset": 0,
    "width": 2048,
    "height": 2048,
```

```
    "format": "BayerRG8",
    "padding": 0,
    "fps": 30.0,
    "loop_images": false,
    "io_timer_interval": 10,
    "max_cached_images_count": 2,
    "wb": {
        "r": 1.0,
        "g1": 1.0,
        "g2": 1.0,
        "b": 1.0,
        "r_off": 0,
        "g_off": 0,
        "b_off": 0
    },
    "filename_template": "{sequence_number:06}.raw",
    "metadata": [
    ]
}
```

parameters

- `cpu_device_id`: CPU device ID
- `directory`: path to target directory
- `offset` (default 0): offset in bytes of image data stored in files
- `width`: width in pixels of images stored in files
- `height`: height in pixels of images stored in files
- `format`: pixel format of images stored in files
- `padding` (default 0): row padding in bytes of images stored in files
- `fps` (default 30.0): desired dispatch FPS
- `loop_images` (default false): dispatch all images from target directory just once or in infinite loop
- `io_timer_interval` (default 10): file I/O status update interval in milliseconds
- `max_cached_images_count` (default 2): maximum number of preloaded images to store in memory, zero means that image is loaded at the time of dispatch
- `wb` (optional): default white balance settings, all parameters are optional (by default gains are set to 1.0 and offsets to 0), green coefficients can be set either together (`g` and `g_off`) or separately (`g1`, `g2`, `g1_off` and `g2_off`, which override `g` and `g_off`), can be modified at runtime
- `filename_template` (optional): string in {fmt} library format to use as filename template, refer to description of this parameter for frames_writer
- `metadata` (optional): metadata as returned by metadata_saver, must be present, if `filename_template` parameter is specified

special parameters

- `total_images` (read only): total number of images found by `raw_frame_player` in the specified `directory`

If both `filename_template` and `metadata` parameters are specified frames are dispatched with recorded metadata except for the following fields:

- white balance settings;

- sequence ID;
- `acquired_at` timestamp;
- `ts` timestamp, if `loop_images` is true.

Note
Hardware trigger mode is not available for `raw_frame_player` component. Common `max_processing_count` parameter is also ignored, `max_cached_images_count` parameter is to be used instead with similar meaning.

4.1.3 `rtsp_source`

Receives data over the network via RTSP (RFC 2326).

JSON configuration:

```
{
    "id": "cam",
    "type": "rtsp_source",
    "dispatch_control_mode": "subscription",
    "cpu_device_id": "cpu_dev",
    "url": "rtsp://192.168.55.1:8554/cam",
    "media_type": "video",
    "transport": "udp",
    "reconnect_delay_sec": 1
}
```

parameters

- `cpu_device_id`: CPU device ID
- `url`: RTSP resource URL
- `media_type` (default ”video”): media type of the stream
- `transport` (default ”udp”): transport protocol for receiving media stream, one of the following values:
  - `tcp`
  - `udp`
- `reconnect_delay_sec` (default 1): time in seconds to wait before trying to reconnect after connection is lost

Note
Common `max_processing_count` and `trigger_mode` parameters along with `trigger` command are ignored by `rtsp_source` component.

4.1.4 `v4l2cam`

Video4Linux2 camera.

JSON configuration:

```
{
    "id": "cam",
    "type": "v4l2cam",
    "max_processing_count": 2,
    "dispatch_control_mode": "subscription",
    "cpu_device_id": "cpu_dev",
    "v4l2_device": "/dev/video0",
    "max_open_retries": -1,
    "wait_after_error_sec": 3,
    "preallocate_buffers": 0,
    "min_buffers_queue_size": 1,
    "sensor_mode": 1,
    "pixel_format": "",
    "width": 3840,
    "height": 2160,
    "custom_params" : [
        { "white_balance_temperature_auto": true },
        { "exposure_auto": 3 }
    ],
    "black_level": 0,
    "exposure": 10000,
    "fps": 60.0,
    "gain": 0.0,
    "wb": {
        "r": 1.0,
        "g1": 1.0,
        "g2": 1.0,
        "b": 1.0,
        "r_off": 0,
        "g_off": 0,
        "b_off": 0
    }
}
```

parameters

- `cpu_device_id`: CPU device ID
- `v4l2_device`: Linux device name corresponding to this camera
- `max_open_retries` (default -1): the maximum number of retries to open the camera before giving up (and transitioning to the `disconnected` state), negative value means unlimited
- `wait_after_error_sec` (default 3): time in seconds between attempts to open the camera
- `preallocate_buffers` (default 0): use `VIDIOC_REQBUFS` to preallocate specified number of buffers if not zero, otherwise use `VIDIOC_CREATE_BUFS` to allocate buffers dynamically
- `min_buffers_queue_size` (default 1): minimum number of buffers kept in the device queue, should be less than `max_processing_count`
- `sensor_mode` (optional): sensor mode for camera, not modified by default
- `pixel_format` (optional): FourCC image format to request when setting camera format, not modified by default
- `width` (optional): frame width to request when setting camera format, not modified by default
- `height` (optional): frame height to request when setting camera format, not modified by default
- `custom_params` (optional): custom camera control parameters, names can be looked up in `v4l2-ctl -l` output

- `black_level` (default 0): sensor black level to use in image metadata (scaled accordingly to output image format bit-depth)
- `exposure` (optional): camera exposure in microseconds, not modified by default
- `fps` (optional): camera FPS limit, not modified by default
- `gain` (optional): camera gain in unspecified units, not modified by default
- `wb` (optional): default white balance settings, all parameters are optional (by default gains are set to 1.0 and offsets to 0), green coefficients can be set either together (`g` and `g_off`) or separately (`g1`, `g2`, `g1_off` and `g2_off`, which override `g` and `g_off`)

No camera controls or parameters (like selected pixel format) are modified unless specified in configuration. They are persistent until reboot or kernel driver reload and can be set using external tools like `v4l2-ctl`. Possible values and combinations of `pixel_format`, `width` and `height` can be looked up in `v4l2-ctl --list-formats-ext` output.

Note
Trigger-related common parameters and command aren't supported by `v4l2_camera` component.

4.1.5 `xicamera`

XIMEA camera.

JSON configuration:

```
{
    "id": "cam",
    "type": "xicamera",
    "max_processing_count": 2,
    "dispatch_control_mode": "subscription",
    "trigger_mode": {
        "mode": "free_run",
        "line": 0
    },
    "cpu_device_id": "cpu_dev",
    "serial_number": "XECAS1930002",
    "debug_level": "WARNING",
    "auto_bandwidth_calculation": true,
    "image_format": "RAW8",
    "switch_red_and_blue": false,
    "max_open_retries": -1,
    "wait_after_error_sec": 3,
    "roi_region": {
        "offset_x": 0,
        "offset_y": 0,
        "width": 1920,
        "height": 1080
    },
    "custom_params": [
        { "bpc": 1 },
        { "column_fpn_correction": 1 },
        { "row_fpn_correction": 1 },
        { "column_black_offset_correction": 1 },
        { "row_black_offset_correction": 1 }
    ],
    "buffer_mode": "safe",
```

```
    "proc_num_threads": 0,
    "image_capture_timeout": 5000,
    "ts_offset": 0,
    "exposure_offset": -1,
    "exposure": 10000,
    "gain": 0.0,
    "fps": 0.0,
    "aperture": 0.0,
    "auto_wb": false,
    "wb": {
        "r": 1.0,
        "g1": 1.0,
        "g2": 1.0,
        "b": 1.0,
        "r_off": 0,
        "g_off": 0,
        "b_off": 0
    }
}
```

parameters

- `cpu_device_id`: CPU device ID
- `serial_number`: serial number of XIMEA camera
- `debug_level` (default ”WARNING”): xiAPI debug level, one of the following values:
  - `DETAIL`
  - `TRACE`
  - `WARNING`
  - `ERROR`
  - `FATAL`
  - `DISABLED`
- `auto_bandwidth_calculation` (default true): whether to enable auto bandwidth calculation in xiAPI
- `image_format` (default ”RAW8”): camera output xiAPI image format
- `switch_red_and_blue` (default false): whether to assume RGB output channel order instead of xiAPI default BGR, should be used together with accordingly set `ccMTX*` parameters in `custom_params` section
- `max_open_retries` (default -1): the maximum number of retries to open the camera before giving up (and transitioning to the `disconnected` state), negative value means unlimited
- `wait_after_error_sec` (default 3): time in seconds between attempts to open the camera
- `roi_region` (optional): camera ROI, by default full frame is used
- `custom_params` (optional): custom camera parameters from xiAPI
- `buffer_mode` (default ”safe”): ”unsafe” setting together with `image_format` set to ”TRANSPORT_DATA” avoids copying the image from xiAPI and returned data pointer is used directly instead
- `proc_num_threads` (default 0): number of threads per image processor (if value is zero or negative auto-detected default is used)
- `image_capture_timeout` (default 5000): get image timeout in milliseconds
- `ts_offset` (default 0): camera timestamp offset, which will be subtracted from reported value
- `exposure_offset` (default -1): correction for reported exposure time, -1 means auto-detect

- `exposure` (default 10000): camera exposure in microseconds, can be modified at runtime
- `gain` (default 0.0): camera gain in dB, can be modified at runtime
- `fps` (default 0.0): camera FPS limit, zero means unlimited (free run), can be modified at runtime
- `aperture` (default 0.0): lens aperture, zero means do not enable lens control, can be modified at runtime
- `auto_wb` (default false): enable xiAPI auto white balance, has no effect if `image_format` is set to `TRANSPORT_DATA`, can be modified at runtime
- `wb` (optional): default white balance settings, all parameters are optional (by default gains are set to 1.0 and offsets to 0), green coefficients can be set either together (`g` and `g_off`) or separately (`g1`, `g2`, `g1_off` and `g2_off`, which override `g` and `g_off`), can be modified at runtime

4.2 Sinks

Components of this kind are the final consumers of data in the processing chain. They have no outputs, but only inputs. Thus, it should be one of the terminal links in the processing chain.

Common parameter for all sinks is:

```
{
    "autostart": false
}
```

- `autostart` (default false): If set to true, sink component will allow data to be dispatched to it as soon as the image parameters are received.

Any sink also supports the following two commands:

- `on`: makes sink start processing images
- `off`: makes sink stop processing images

See `iff_execute()` for more details on command execution by elements.

4.2.1 `awb_aec`

Sets white balance and exposure based on the image histogram.

JSON configuration:

```
{
    "id": "ctrl",
    "type": "awb_aec",
    "max_processing_count": 2,
    "autostart": false,
    "cpu_device_id": "cpu_dev",
    "aec_enabled": true,
    "awb_enabled": true,
    "noise_floor": 0.01,
    "saturation": 0.987,
    "min_area": 0.01,
    "wb_stretch": false,
```

```
    "wb_ratio_under": 0.0,
    "wb_ratio_over": 1.0,
    "wb_margin_under": 0.0,
    "wb_margin_over": 0.0,
    "wb_comp_min": 0.0,
    "wb_comp_max": 1.0,
    "wait_limit": 3,
    "add_frames": 0,
    "min_exposure": 100,
    "max_exposure": 16000,
    "exposure_margin": 0.05,
    "hdr_threshold_low": 1.0,
    "hdr_threshold_high": 1.0,
    "ev_correction": 0.0,
    "hdr_median_ev": -3.0
}
```

formulas

$whitepoint = bins - 1$

where $bins$ is number of bins in input histogram

$total_i = \sum_j in_{ij}$

$sum_i = \sum_j in_{ij} \cdot j$

$i \in \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$ or $i \in \{\mathrm{R}, \mathrm{G1}, \mathrm{G2}, \mathrm{B}\}$ depending on input histogram format

$j \in \mathrm{I}$

$\mathrm{I} = \{0, 1, 2, \ldots, whitepoint\}$

$m = \arg\max \frac{sum_i}{total_i}$ (a)

(a) selects color channel with the highest mean value.

formula for simple white balance

The most simple approach to auto white balance is to scale each color channel so that their mean values match (it works well when so called gray world assumption holds). For that the most bright (with the highest mean value) channel is left unscaled and calculated gains are applied to the remaining ones.

$threshold = saturation \cdot (whitepoint - black\_level) + black\_level$

where $black\_level$ is taken from input histogram metadata

$saturated_i = \sum_{j \geq threshold} in_{ij}$

$green\_factor_i = \begin{cases} 2 & i = G \\ 1 & \text{otherwise} \end{cases}$

$sat\_cnt = \max \frac{saturated_i}{green\_factor_i}$

$\mathrm{O}_i = \{x \in \mathrm{I} \mid \sum_{j \geq x} in_{ij} \leq sat\_cnt \cdot green\_factor_i\} \cup \{bins\}$

$$cut_i = \min_{x \in \mathrm{O}_i} x$$

$$cnt\_cut_i = \sum_{j \geq cut_i} in_{ij}$$

$$corr_i = (sat\_cnt \cdot green\_factor_i - cnt\_cut_i) \cdot (cut_i - 1) + \sum_{j \geq cut_i} in_{ij} \cdot j$$

$$sum\_corr_i = sum_i - corr_i$$

$$total\_corr_i = total_i - sat\_cnt \cdot green\_factor_i$$

$$m\_corr = \arg\max \frac{sum\_corr_i}{total\_corr_i}$$

$$noise\_level = \min \frac{sum\_corr_i}{total\_corr_i} - black\_level$$

$$out\_off_i = 0$$

$$out\_gain_i = \begin{cases} in\_gain_i & total_R - sat\_cnt \leq min\_area \cdot total_R \\ in\_gain_i & noise\_level \leq noise\_floor \cdot (whitepoint - black\_level) \\ \frac{\frac{sum\_corr_{m\_corr}}{total\_corr_{m\_corr}} - black\_level}{\frac{sum\_corr_i}{total\_corr_i} - black\_level} & \text{otherwise} \end{cases}$$

formula for histogram stretch white balance

This is custom auto white balance algorithm aimed at better quality video encoding for streaming of hazy images and to be reverted on the receiving end.

$$comp\_range = wb\_comp\_max - wb\_comp\_min$$

$$q\_under_i = \min_{x \in \Upsilon_i} x$$

$$\Upsilon_i = \{x \in \mathrm{I} \mid \sum_{j \leq x} in_{ij} \geq wb\_ratio\_under \cdot total_i\}$$

$$q\_over_i = \min_{x \in \mathrm{O}_i} x$$

$$\mathrm{O}_i = \{x \in \mathrm{I} \mid \sum_{j \leq x} in_{ij} > wb\_ratio\_over \cdot total_i\} \cup \{whitepoint\}$$

$$range_i = q\_over_i - q\_under_i$$

$$cut\_under_i = \frac{\lfloor q\_under_i - range_i \cdot wb\_margin\_under \rfloor}{whitepoint}$$

$$cut\_over_i = \frac{\lfloor q\_over_i + range_i \cdot wb\_margin\_over \rfloor}{whitepoint}$$

$$out\_off_i = \begin{cases} cut\_under_i & cut\_under_i \geq 0 \\ 0 & cut\_under_i < 0 \end{cases}$$

$$out\_gain_i = \begin{cases} \frac{comp\_range}{cut\_over_i - out\_off_i} & cut\_over_i \leq 1 \\ \frac{comp\_range}{1 - out\_off_i} & cut\_over_i > 1 \end{cases}$$

formula for exposure

For exposure calculation only channel with the highest current mean value is evaluated. Either median or mean value is taken (depending on chosen algorithm mode, which can be switched automatically by comparing how much these values differ) and compared to target value. Exposure correction factor is calculated from average of these two values and then applied to current exposure to get new exposure setting.

$$middle_i = \min_{x \in \mathrm{M}_i} x \qquad \text{(b)}$$

$$\mathrm{M}_i = \{x \in \mathrm{I} \mid \sum_{j \le x} in_{ij} > \frac{total_i}{2}\} \qquad \text{(c)}$$

$$median = \frac{middle_m}{whitepoint} \qquad \text{(d)}$$

$$mean = \frac{sum_m}{total_m \cdot whitepoint} \qquad \text{(e)}$$

$$target\_mean = 2^{ev\_correction-1} \qquad \text{(f)}$$

$$target\_median = 2^{hdr\_median\_ev} \qquad \text{(g)}$$

$$hdr\_diff = \begin{cases} hdr\_threshold\_high & \frac{mean-median}{mean} > hdr\_diff \text{ for previous image} \\ hdr\_threshold\_low & \text{otherwise} \end{cases} \qquad \text{(h)}$$

$$target\_exp = exposure \cdot \begin{cases} \frac{mean+target\_mean}{2 \cdot mean} & \frac{mean-median}{mean} \le hdr\_diff \\ \frac{median+target\_median}{2 \cdot median} & \frac{mean-median}{mean} > hdr\_diff \end{cases} \qquad \text{(i)}$$

where $exposure$ is exposure time taken from image metadata

$$set\_exp = \begin{cases} min\_exposure & target\_exp < min\_exposure \\ target\_exp & min\_exposure \le target\_exp \le max\_exposure \\ max\_exposure & max\_exposure < target\_exp \end{cases} \qquad \text{(j)}$$

$$out\_exp = \begin{cases} set\_exp & \frac{set\_exp-exposure}{exposure} \le -exposure\_margin \\ exposure & -exposure\_margin < \frac{set\_exp-exposure}{exposure} < exposure\_margin \\ set\_exp & exposure\_margin \le \frac{set\_exp-exposure}{exposure} \end{cases} \qquad \text{(k)}$$

(b)-(c) defines non-normalized median value. (d)-(e) defines normalized mean and median values. (f)-(g) defines target mean and median values. (h)-(i) selects auto-exposure mode and applies it to get target exposure time. (j) clamps calculated value to the defined boundaries. (k) checks if target exposure falls within specified margins from current setting and discards an update in that case.

parameters

- `cpu_device_id`: CPU device ID
- `aec_enabled` (default false): enable/disable exposure calculation and control, can be modified at runtime
- `awb_enabled` (default false): enable/disable white balance calculation and control, can be modified at runtime
- `noise_floor` (default 0.01): normalized noise floor (affects simple white balance calculation)
- `saturation` (default 0.987): if normalized pixel value is above this threshold it is considered saturated (affects simple white balance calculation)
- `min_area` (default 0.01): minimal non-saturated area (1.0 being whole image) required to trigger simple white balance calculation
- `wb_stretch` (default false): enables histogram stretch white balance algorithm instead of simple one
- `wb_ratio_under` (default 0.0): percentile for shadow compression section in histogram stretch white balance
- `wb_ratio_over` (default 1.0): percentile for highlights compression section in histogram stretch white balance
- `wb_margin_under` (default 0.0): relative margin for shadow compression section in histogram stretch white balance

- `wb_margin_over` (default 0.0): relative margin for highlights compression section in histogram stretch white balance
- `wb_comp_min` (default 0.0): maximum normalized value for shadow compression section in histogram stretch white balance
- `wb_comp_max` (default 1.0): minimum normalized value for highlights compression section in histogram stretch white balance
- `wait_limit` (default 3): how many frames to wait for exposure to change in image metadata before assuming that it's stuck and continuing to try to set exposure value
- `add_frames` (default 0): allows to accumulate a histogram from several frames, useful in case of flickering image e.g. due to artificial lighting
- `min_exposure` (default 100): minimum exposure time in us that is going to be set
- `max_exposure` (default 0): maximum exposure time in us that is going to be set
- `exposure_margin` (default 0.05): do not adjust the exposure if relative change is less than this value
- `hdr_threshold_low` (default 1.0 meaning HDR mode disabled): switch to LDR mode if median value is not bigger than mean value by that relative to mean value amount
- `hdr_threshold_high` (default 1.0 meaning HDR mode disabled): switch to HDR mode if median value is bigger than mean value by that relative to mean value amount
- `ev_correction` (default 0.0): correction in EV stops of target mean value compared to 50% (-1 EV) for LDR mode
- `hdr_median_ev` (default -3.0): target median value in EV stops (0 EV is white point) for HDR mode

Use value 1.0 for `hdr_threshold_low` and `hdr_threshold_high` to disable HDR mode, and -65536.0 to disable LDR mode.

callbacks

- `wb_callback`: this callback is called when white balance parameters have been calculated by the element
- `exposure_callback`: this callback is called when exposure and gain parameters have been calculated by the element

wb_callback data format:

```
{
    "wb": {
        "r": 1.0,
        "g1": 1.0,
        "g2": 1.0,
        "b": 1.0,
        "r_off": 0.0,
        "g1_off": 0.0,
        "g2_off": 0.0,
        "b_off": 0.0
    }
}
```

exposure_callback data format:

```
{
    "exposure": 10000,
    "gain": 1.0
}
```

4.2.2 `files_writer`

Writes all received frames to a given file in the given directory until stopped or end-of-stream event is received. Each time start command is received by writer it begins a new file.

JSON configuration:

```
{
    "id": "writer",
    "type": "files_writer",
    "max_processing_count": 2,
    "autostart": false,
    "cpu_device_id": "cpu_dev",
    "write_directory": "saved_files",
    "direct_io": true,
    "io_timer_interval": 10
}
```

parameters

- `cpu_device_id`: CPU device ID
- `write_directory` (default ”saved_files”): path to the directory to save files in
- `direct_io` (default true): whether to use direct I/O (`O_DIRECT` on Linux, `FILE_FLAG_NO_BUFFERING | FILE_FLAG_WRITE_THROUGH` on Windows, `F_NOCACHE` on macOS)
- `io_timer_interval` (default 10): file I/O status update interval in milliseconds

commands

- `on`: takes the following parameters:
  - `filename`: name of the file to write, ISO 8601 time stamp is used by default (if this parameter is empty or omitted)

4.2.3 `frame_exporter`

Dispatches each received buffer to an external consumer via the assigned callback (see `iff_set_export_cal` Dispatch is carried out from a separate thread. It should be used to pass frame data across IFF SDK library boundaries.

JSON configuration:

```
{
    "id": "exporter",
    "type": "frame_exporter",
    "max_processing_count": 2,
    "autostart": false,
    "device_id": "cuda_dev"
}
```

parameters

- `device_id`: Device ID

4.2.4 frames_writer

Writes each received frame to a separate file in the given directory.

JSON configuration:

```
{
    "id": "writer",
    "type": "frames_writer",
    "max_processing_count": 2,
    "autostart": false,
    "cpu_device_id": "cpu_dev",
    "base_directory": "saved_frames",
    "direct_io": true,
    "filename_template": "{sequence_number:06}.raw",
    "io_timer_interval": 10
}
```

parameters

- `cpu_device_id`: CPU device ID
- `base_directory` (default ”saved_frames”): path to the directory to save files in
- `direct_io` (default true): whether to use direct I/O (`O_DIRECT` on Linux, `FILE_FLAG_NO_BUFFERING | FILE_FLAG_WRITE_THROUGH` on Windows, `F_NOCACHE` on macOS)
- `filename_template` (default ”{sequence_number:06}.raw”): string in {fmt} library format to use as filename template. Each `{param_name}` is a name of corresponding frame metadata field. Possible parameter names are:
  - `sequence_number` - frame sequence number for current recording session
  - `padding` - frame data padding
  - `format` - frame pixel format
  - `width` - frame width
  - `height` - frame height
  - `offset_x` - frame horizontal offset
  - `offset_y` - frame vertical offset
  - `cam_ts` - frame timestamp in micro-seconds delivered by camera
  - `utc_time` - frame NTP UTC date and time in ISO 8601 format
  - `black_level` - frame black level
  - `exposure` - frame exposure time
  - `gain` - frame gain
  - `sequence_id` - frame sequence id
- `io_timer_interval` (default 10): file I/O status update interval in milliseconds

special parameters

`frames_writer` has one additional read only parameter:

- `data_offset`: offset in bytes (metadata header size) where image data starts in recorded file

commands

- `on`: takes the following parameters:
  - `subdirectory` (default ””): directory to append to `base_directory`
  - `frames_count` (default 0): maximum number of frames to write, zero means no limit

4.2.5 dng_writer

Writes each received image to a separate uncompressed DNG file in the given directory. Creates the following outputs for each of supported input formats:

- `Mono` - LinearRaw DNG
- `Bayer` - CFA DNG
- `RGB` - RGB TIFF
- `BGR` - RGB TIFF with switched blue and red channels
- `RGBA` - RGB TIFF with alpha channel (not well supported)
- `BGRA` - RGB TIFF with alpha channel and switched blue and red channels
- `Monop` - non-standard LinearRaw DNG with Compression set to 65042
- `Bayerp` - non-standard CFA DNG with Compression set to 65042

JSON configuration:

```
{
    "id": "writer",
    "type": "dng_writer",
    "max_processing_count": 2,
    "autostart": false,
    "cpu_device_id": "cpu_dev",
    "base_directory": "saved_frames",
    "io_timer_interval": 10,
    "filename_template": "{sequence_number:06}.raw",
    "make": "",
    "model": "",
    "serial_number": "",
    "copyright": "",
    "description": "",
    "frame_rate": 0.0,
    "color_profile": {
        "CalibrationIlluminant1": "D50",
        "ColorMatrix1": [
             3.1338561, -1.6168667, -0.4906146,
            -0.9787684,  1.9161415,  0.0334540,
             0.0719453, -0.2289914,  1.4052427
        ]
    },
    "dcp_file": ""
}
```

parameters

All frames_writer parameters are supported with an addition of:

- `make` (default ””): string, that will be written to Make TIFF tag and UniqueCameraModel DNG tag
- `model` (default ””): string, that will be written to Model TIFF tag and UniqueCameraModel DNG tag
- `serial_number` (default ””): string, that will be written to CameraSerialNumber DNG tag, if not empty

- `copyright` (default ””): string, that will be written to Copyright TIFF tag
- `description` (default ””): string, that will be written to ImageDescription TIFF tag
- `frame_rate` (default 0.0): rational number, that will be written to FrameRate CinemaDNG tag, if not zero
- `color_profile` (optional): DNG color profile, that will be embedded into the file in case of `Bayer` image format, with the following supported DNG tags:
  - `CalibrationIlluminant1` (default ”D50”) - can be specified as an integer number or as one of the following strings:
    - ”Unknown” (0)
    - ”Daylight” (1)
    - ”Fluorescent” (2)
    - ”Tungsten” (3)
    - ”Flash” (4)
    - ”FineWeather” (9)
    - ”Cloudy” (10)
    - ”Shade” (11)
    - ”DaylightFluorescent” (12)
    - ”DayWhiteFluorescent” (13)
    - ”CoolWhiteFluorescent” (14)
    - ”WhiteFluorescent” (15)
    - ”StandardLightA” (17)
    - ”StandardLightB” (18)
    - ”StandardLightC” (19)
    - ”D55” (20)
    - ”D65” (21)
    - ”D75” (22)
    - ”D50” (23)
    - ”ISOStudioTungsten” (24)
    - ”Other” (255)
  - `ColorMatrix1` (default XYZ D50 to sRGB matrix) - 3x3 matrix of floats
- `dcp_file` (optional): path to DNG color profile file, with the following DNG tags used from it for the output files in case of `Bayer` image format:
  - `BaselineExposureOffset` - SRATIONAL tag type is written (and allowed in input) instead of stated in DNG specification RATIONAL type (which is also accepted in input), since value can be negative
  - `CalibrationIlluminant1` - takes precedence over the one specified in `color_profile` parameter
  - `CalibrationIlluminant2`
  - `ColorMatrix1` - takes precedence over the one specified in `color_profile` parameter
  - `ColorMatrix2`
  - `DefaultBlackRender`
  - `ForwardMatrix1`
  - `ForwardMatrix2`
  - `ProfileCalibrationSignature`
  - `ProfileCopyright`
  - `ProfileEmbedPolicy`
  - `ProfileHueSatMapData1`
  - `ProfileHueSatMapData2`
  - `ProfileHueSatMapDims`

- `ProfileHueSatMapEncoding`
- `ProfileLookTableData`
- `ProfileLookTableDims`
- `ProfileLookTableEncoding`
- `ProfileName`
- `ProfileToneCurve`
- `UniqueCameraModel` - value is compared to UniqueCameraModel DNG tag generated from `make` and `model` parameters and a warning is issued in case of mismatch

Other metadata tags, like white balance (AsShotNeutral), are filled from image metadata.

references

- TIFF 6.0 Specification
- TIFF/EP Specification
- DNG Specification (version 1.5.0.0)
- CinemaDNG Image Data Format Specification (version 1.1.0.0)

4.2.6 `metadata_saver`

Saves metadata of received images to an internal buffer, which can be accessed externally.

JSON configuration:

```
{
    "id": "metadata",
    "type": "metadata_saver",
    "max_processing_count": 2,
    "autostart": false,
    "cache_size": 4096
}
```

parameters

- `cache_size` (default 4096): maximum metadata buffer size in number of frames

Older information gets dropped when number of images for which metadata was saved exceeds `cache_size` limit.

special parameters

- `metadata` (read only): saved metadata can be read by getting the value of this parameter

4.2.7 `rtsp_stream`

Represents an RTSP video stream. Automates creation and configuration of RTSP resources within RTSP streaming server.

JSON configuration:

```
{
    "id": "netstream",
    "type": "rtsp_stream",
    "relative_uri": "/cam",
    "name": "netstream"
}
```

parameters

- `relative_uri`: relative URI of an RTSP resource within RTSP server
- `name` (optional): name of the stream, set directly to the **`a=control:`** attribute of resource SDP (if this parameter is not specified component `id` will be used as a name)

Note
Common `max_processing_count` and `autostart` parameters along with `on` and `off` commands are ignored by `rtsp_stream` component. Image processing is instead automatically controlled by RTSP server itself based on RTSP client requests.

4.3 Filters

Filters are components that have inputs and outputs. They can not be neither initial nor terminal link of the processing chain. Filters can analyze, alter or pass through as is their input frames stream.

4.3.1 `averager`

Averages specified number of input images.

JSON configuration:

```
{
    "id": "avg",
    "type": "averager",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
    "num_frames": 1
}
```

formula

$$out = \frac{1}{num\_frames} \cdot \sum_i in_i$$

$$i \in \{1, 2, \ldots, num\_frames\}$$

parameters

- `cpu_device_id`: CPU device ID
- `num_frames` (default 1): number of images to average

Filter outputs one image per `num_frames` input images taking metadata from the first frame in sequence.

4.3.2 `decoder`

Decodes incoming video stream.

JSON configuration:

```
{
    "id": "nvdec",
    "type": "decoder",
    "max_processing_count": 2,
    "decoder_type": "nvidia",
    "cpu_device_id": "cpu_dev",
    "gpu_device_id": "cuda_dev"
}
```

parameters

- `decoder_type`: type of decoder library, must be `nvidia` (only NVIDIA hardware decoder is supported by IFF now)
- `cpu_device_id`: CPU device ID
- `gpu_device_id`: GPU device ID

4.3.3 `encoder`

Encodes the image.

JSON configuration:

```
{
    "id": "nvenc",
    "type": "encoder",
    "max_processing_count": 2,
    "encoder_type": "nvidia",
    "cpu_device_id": "cpu_dev",
    "gpu_device_id": "cuda_dev",
    "codec": "H264",
    "profile": "H264_HIGH",
    "level": "H264_51",
    "config_preset": "DEFAULT",
    "preset_tuning": "ULTRA_LOW_LATENCY",
    "multipass": "DISABLED",
    "rc_mode": "CBR",
    "fps": 30.0,
    "bitrate": 30000000,
    "max_bitrate": 40000000,
    "idr_interval": 30,
    "iframe_interval": 30,
    "repeat_spspps": true,
    "virtual_buffer_size": 0,
    "slice_intrarefresh_interval": 0,
    "qp": 28,
    "min_qp_i": -1,
    "max_qp_i": -1,
    "min_qp_p": -1,
    "max_qp_p": -1,
    "report_metadata": false,
    "max_performance": false
}
```

parameters

- `encoder_type`: type of encoder library, must be `nvidia` (only NVIDIA hardware encoder is supported by IFF now)
- `cpu_device_id`: CPU device ID
- `gpu_device_id`: GPU device ID
- `codec`: video codec to use (should be ”H264” or ”H265”)
- `profile` (default ”H264_HIGH”, or ”H265_MAIN”, or ”H265_MAIN10”): codec profile, one of the following values:
  - for H264 codec:
    - `H264_MAIN`
    - `H264_BASELINE`
    - `H264_HIGH`
  - for H265 codec:
    - `H265_MAIN`
    - `H265_MAIN10`
- `level` (default ”H264_51” or ”H265_62_HIGH_TIER”): codec level, one of the following values:
  - for H264 codec:
    - `H264_1`
    - `H264_1b`
    - `H264_11`
    - `H264_12`
    - `H264_13`
    - `H264_2`
    - `H264_21`
    - `H264_22`
    - `H264_3`
    - `H264_31`
    - `H264_32`
    - `H264_4`
    - `H264_41`
    - `H264_42`
    - `H264_5`
    - `H264_51`
    - `H264_52`
    - `H264_60`
    - `H264_61`
    - `H264_62`
  - for H265 codec:
    - `H265_1_MAIN_TIER`
    - `H265_2_MAIN_TIER`
    - `H265_21_MAIN_TIER`
    - `H265_3_MAIN_TIER`
    - `H265_31_MAIN_TIER`
    - `H265_4_MAIN_TIER`
    - `H265_41_MAIN_TIER`
    - `H265_5_MAIN_TIER`
    - `H265_51_MAIN_TIER`
    - `H265_52_MAIN_TIER`

    * H265_6_MAIN_TIER
    * H265_61_MAIN_TIER
    * H265_62_MAIN_TIER
    * H265_1_HIGH_TIER
    * H265_2_HIGH_TIER
    * H265_21_HIGH_TIER
    * H265_3_HIGH_TIER
    * H265_31_HIGH_TIER
    * H265_4_HIGH_TIER
    * H265_41_HIGH_TIER
    * H265_5_HIGH_TIER
    * H265_51_HIGH_TIER
    * H265_52_HIGH_TIER
    * H265_6_HIGH_TIER
    * H265_61_HIGH_TIER
    * H265_62_HIGH_TIER
- `config_preset` (default ”DEFAULT”): encoding preset. Can be used one of the following presets:
  - on Jetson:
    * `TEGRA_DISABLE` - ”Disabled” encoder hardware preset
    * `TEGRA_ULTRAFAST` or `DEFAULT` - encoder hardware preset with ”Ultra-Fast” per frame encode time
    * `TEGRA_FAST` - encoder hardware preset with ”Fast” per frame encode time
    * `TEGRA_MEDIUM` - encoder hardware preset with ”Medium” per frame encode time
    * `TEGRA_SLOW` - encoder hardware preset with ”Slow” per frame encode time
  - on desktop GPU (performance degrades and quality improves as we move from P1 to P7):
    * `P1` or `DEFAULT` - default preset
    * `P2`
    * `P3`
    * `P4`
    * `P5`
    * `P6`
    * `P7`
- `preset_tuning` (default ”ULTRA_LOW_LATENCY”): preset tuning mode. Supported on desktop GPU only. Following modes are supported:
  - `LOSSLESS` - tune presets for lossless encoding
  - `HIGH_QUALITY` - tune presets for latency tolerant encoding
  - `LOW_LATENCY` - tune presets for low latency streaming
  - `ULTRA_LOW_LATENCY` - tune presets for ultra low latency streaming
- `multipass` (default ”DISABLED”): multi pass encoding mode. Supported on desktop GPU only. Following modes are supported:
  - `DISABLED` - single pass mode
  - `QUARTER_RESOLUTION` - two pass encoding is enabled where first pass is quarter resolution
  - `FULL_RESOLUTION` - two pass encoding is enabled where first pass is full resolution
- `rc_mode` (default ”CBR”): rate control mode. Following modes are supported:
  - on both Jetson and desktop GPU:
    * `VBR` - variable bit-rate mode
    * `CBR` - constant bit-rate mode
  - on desktop GPU only:

    * `CONSTQP` - constant QP mode
- `fps`(default 30.0): encoder fps, can be modified at runtime
- `bitrate` (default 4194304): stream bit-rate in bps, can be modified at runtime
- `max_bitrate` (optional): maximum stream bit-rate, used for VBR mode only
- `idr_interval` (default 30): IDR frame interval
- `iframe_interval` (default 30): I frame interval
- `repeat_spspps` (default true): whether to attach SPS/PPS/VPS to each IDR frame, otherwise they are attached only to the first one
- `virtual_buffer_size` (default 0): specifies the VBV/HRD buffer size in bits, set 0 to use the default buffer size
- `slice_intrarefresh_interval` (default 0): specify the encoder slice intra refresh interval
- `qp` (default 28): specifies QP to be used for encoding
- `min_qp_i`: min QP for I-frames
- `max_qp_i`: max QP for I-frames
- `min_qp_p`: min QP for P-frames
- `max_qp_p`: max QP for P-frames
- `report_metadata` (default false): if set to `true` encoder will output metadata with every encoded frame
- `max_performance` (default false): for Jetson only, set to `true` to enable maximum performance

commands

- `force_idr`: forces next incoming image to be encoded as an IDR frame, takes no parameters

4.3.4 `frame_dropper`

Drops frames in the repeating pattern: pass N frames, drop M frames.

JSON configuration:

```
{
    "id": "drop",
    "type": "frame_dropper",
    "max_processing_count": 2,
    "dispatch_count": 1,
    "drop_count": 1
}
```

parameters

- `dispatch_count` (default 1): how many frames to pass-through at the beginning of the pattern
- `drop_count` (default 1): how many frames to drop at the end of the pattern

`dispatch_count` / (`dispatch_count` + `drop_count`) gives the percentage of passed-through frames and consequently the FPS change factor.

4.3.5 `histogram`

Builds a histogram for Bayer or mono image (depth 8 to 16).

JSON configuration:

```
{
    "id": "hist",
    "type": "histogram",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
    "bins": 256
}
```

formula

$whitepoint = bins - 1$

$$out_{xy} = \sum_{(i,j)\in\Pi_y} \mathrm{I}_x(in_{ij})$$

$x \in \{0, 1, 2, \ldots, whitepoint\}$

$y \in \mathrm{X}$

$$\mathrm{I}_x(z) = \begin{cases} 0 & \frac{z}{white\_level} < \frac{x}{whitepoint} \\ 1 & \frac{x}{whitepoint} \le \frac{z}{white\_level} < \frac{x+1}{whitepoint} \\ 0 & \frac{x+1}{whitepoint} \le \frac{z}{white\_level} \end{cases} \qquad \text{(a)}$$

$$\Pi_y = \{(i,j) \mid i,j \in \mathbb{N}_0, i < w, j < h, (i + c\_x) \bmod 2 + 2 \cdot ((j + c\_y) \bmod 2) \in \Upsilon_y\} \qquad \text{(b)}$$

where (w, h) are image dimensions, and (c_x, c_y) defines image Bayer pattern shift compared to RGGB

$\Upsilon_{\mathrm{V}} = \{0,1,2,3\}, \Upsilon_{\mathrm{R}} = \{0\}, \Upsilon_{\mathrm{G}} = \{1,2\}, \Upsilon_{\mathrm{B}} = \{3\}$

(a) defines whether value z falls into bin x. (b) defines pixel positions for specific color channel from X.

parameters

- `cpu_device_id`: CPU device ID
- `bins` (default 256): bin count for histogram (should be a power of 2, from 256 to 65536)

Output format is one of the following:

- `HistogramMono<bins>Int` (`Mono` input image format) - $\mathrm{X} = \{\mathrm{V}\}$
- `Histogram3Bayer<bins>Int` (`Bayer` input image format) - $\mathrm{X} = \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$

4.3.6 `image_crop`

Crops the image.

JSON configuration:

```
{
    "id": "crop",
    "type": "image_crop",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
```

```
    "offset_x": 0,
    "offset_y": 0,
    "width": 1024,
    "height": 1024
}
```

parameters

- `cpu_device_id`: CPU device ID
- `offset_x`, `offset_y` (default 0): coordinates of top left corner of crop area, input image width/height is added to the value if it is negative
- `width`, `height` (default 0): dimensions of crop area, input image width/height is added to the value if it is non-positive

By default this filter just copies input image to output buffer, which could be used to get rid of a row padding.

4.3.7 `metadata_exporter`

Exports metadata of every frame passed through it using `new_frame_metadata` callback

JSON configuration:

```
{
    "id": "metadata",
    "type": "metadata_exporter",
    "static_metadata": {
        "ip": "127.0.0.1"
    }
}
```

parameters

- `static_metadata`: any static metadata defined by user, this metadata will be added to the metadata of each frame

callbacks

- `new_frame_metadata`: this callback returns the frame's metadata and is called when the frame passes through the filter

4.3.8 `sub_monitor`

Passes through any incoming images while providing callbacks on pipeline status change events.

JSON configuration:

```
{
    "id": "sub_mon",
    "type": "sub_monitor"
}
```

callbacks

- `on_new_consumer`: this callback is called when some connection to output of this element becomes active (images begin to flow)

4.3.9 xiprocessor

Processes images using xiAPI offline processing.

JSON configuration:

```
{
    "id": "xiproc",
    "type": "xiprocessor",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
    "custom_params": [
        { "gammaY": 0.47 }
    ],
    "image_format": "RGB32",
    "switch_red_and_blue": false,
    "proc_num_threads": 0
}
```

parameters

- `cpu_device_id`: CPU device ID
- `custom_params` (optional): custom parameters from xiAPI
- `image_format` (default "RGB32"): output xiAPI image data format
- `switch_red_and_blue` (default false): whether to assume RGB output channel order instead of xiAPI default BGR, should be used together with accordingly set `ccMTX*` parameters in `custom_params` section
- `proc_num_threads` (default 0): number of threads per image processor (if value is zero or negative auto-detected default is used)

Set `image_format` to `RAW16` for just unpacking of packed transport data format or use default `RGB32` setting for full processing including demosaicing.

4.3.10 cuda_processor

Processes incoming images on NVIDIA GPU. This filter can perform different processing operations on image. Those operations can be arranged into a pipeline.

JSON configuration:

```
{
    "id": "gpuproc",
    "type": "cuda_processor",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
    "gpu_device_id": "cuda_dev",
    "elements": [
        { "id": "import_from_host", "type": "import_from_host" },
        { "id": "black_level",      "type": "black_level" },
        { "id": "white_balance",    "type": "white_balance" },
        { "id": "demosaic",         "type": "demosaic",         "algorithm": "HQLI " },
        { "id": "color_correction", "type": "color_correction", "matrix": [ 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0 ] },
        { "id": "gamma",            "type": "gamma8",           "linear": 0.018, "power": 0.45 },
```

```
        { "id": "export_to_device", "type": "export_to_device", "output_format": "
            NV12_BT709",            "output_name": "yuv" },
        { "id": "hist",             "type": "histogram",        "output_format": "
            Histogram4Bayer256Int", "output_name": "histogram" }
    ],
    "connections": [
        { "src": "import_from_host", "dst": "black_level" },
        { "src": "black_level",      "dst": "white_balance" },
        { "src": "white_balance",    "dst": "demosaic" },
        { "src": "demosaic",         "dst": "color_correction" },
        { "src": "color_correction", "dst": "gamma" },
        { "src": "gamma",            "dst": "export_to_device" },
        { "src": "black_level",      "dst": "hist" }
    ]
}
```

parameters

- `cpu_device_id`: CPU device ID
- `gpu_device_id`: CUDA device ID
- `elements`: list of required `cuda_processor` pipeline elements (see section below)
  - `id`: unique element ID
  - `type`: element type (see section below for possible values)
- `connections`: list of edges which connect elements into pipeline
  - `src`: element ID used as a source of the connection
  - `dst`: element ID used as a destination of the connection

CUDA processor elements

Import adapters

`import_from_host`
Copies data from CPU buffer taking row pitch into account and unpacking in case of `Mono12p` and `BayerXX12p` formats. It's faster if buffer is CUDA-allocated (page-locked).

JSON configuration

```
{
    "id": "import_from_host",
    "type": "import_from_host"
}
```

`import_from_device`
Copies data from CUDA device buffer taking row pitch into account and unpacking in case of `Mono12p` and `BayerXX12p` formats.

JSON configuration:

```
{
    "id": "import_from_device",
    "type": "import_from_device"
}
```

Export adapters

---

export_to_host
Copies data without conversion to CPU buffer taking row pitch into account. It's faster if buffer is CUDA-allocated (page-locked).

JSON configuration:

```
{
    "id": "export_to_host",
    "type": "export_to_host",
    "output_format": "Mono8",
    "output_name": "export_to_host_out"
}
```

---

export_to_devmem
Copies data without conversion to CUDA device buffer taking row pitch into account.

JSON configuration:

```
{
    "id": "export_to_devmem",
    "type": "export_to_devmem",
    "output_format": "RGB8",
    "output_name": "export_to_devmem_out"
}
```

---

export_to_device
Copies data to CUDA device buffer converting to specified format. Rows are aligned to 4 byte boundaries.

JSON configuration:

```
{
    "id": "export_to_device",
    "type": "export_to_device",
    "output_format": "YV12_BT709",
    "output_name": "export_to_device_out"
}
```

formula for YUV conversion

$Y' = K_R \cdot R' + (1 - K_R - K_B) \cdot G' + K_B \cdot B'$

$P'_B = \frac{1}{2} \cdot \frac{B' - Y'}{1 - K_B}$

$P'_R = \frac{1}{2} \cdot \frac{R' - Y'}{1 - K_R}$

where R', G', B' are normalized to [0, 1]

for n-bit full range:

$Y = 255 \cdot Y' \cdot 2^{n-8}$

$C_B = (255 \cdot P'_B + 128) \cdot 2^{n-8}$

$C_R = (255 \cdot P'_R + 128) \cdot 2^{n-8}$

or in matrix form

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \left(\begin{pmatrix} K_R & 1 - K_R - K_B & K_B \\ \frac{1}{2} \cdot \frac{K_R}{K_B - 1} & \frac{1}{2} \cdot \frac{1 - K_R - K_B}{K_B - 1} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \cdot \frac{1 - K_R - K_B}{K_R - 1} & \frac{1}{2} \cdot \frac{K_B}{K_R - 1} \end{pmatrix} \cdot \begin{pmatrix} 255 \cdot R' \\ 255 \cdot G' \\ 255 \cdot B' \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \right) \cdot 2^{n-8}$$

for n-bit limited range:

$Y = (219 \cdot Y' + 16) \cdot 2^{n-8}$

$C_B = (224 \cdot P'_B + 128) \cdot 2^{n-8}$

$C_R = (224 \cdot P'_R + 128) \cdot 2^{n-8}$

or in matrix form

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \left(\begin{pmatrix} \frac{219}{255} \cdot K_R & \frac{219}{255} \cdot (1 - K_R - K_B) & \frac{219}{255} \cdot K_B \\ \frac{224}{255} \cdot \frac{1}{2} \cdot \frac{K_R}{K_B - 1} & \frac{224}{255} \cdot \frac{1}{2} \cdot \frac{1 - K_R - K_B}{K_B - 1} & \frac{224}{255} \cdot \frac{1}{2} \\ \frac{224}{255} \cdot \frac{1}{2} & \frac{224}{255} \cdot \frac{1}{2} \cdot \frac{1 - K_R - K_B}{K_R - 1} & \frac{224}{255} \cdot \frac{1}{2} \cdot \frac{K_B}{K_R - 1} \end{pmatrix} \cdot \begin{pmatrix} 255 \cdot R' \\ 255 \cdot G' \\ 255 \cdot B' \end{pmatrix} + \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} \right) \cdot 2^{n-8}$$

- **BT.601**

$K_R = 0.299$

$K_B = 0.114$

for n-bit full range

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \left(\begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} \cdot \begin{pmatrix} 255 \cdot R' \\ 255 \cdot G' \\ 255 \cdot B' \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \right) \cdot 2^{n-8}$$

for n-bit limited range

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \left(\begin{pmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{pmatrix} \cdot \begin{pmatrix} 255 \cdot R' \\ 255 \cdot G' \\ 255 \cdot B' \end{pmatrix} + \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} \right) \cdot 2^{n-8}$$

- **BT.709**

$K_R = 0.2126$

$K_B = 0.0722$

for n-bit full range

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \left(\begin{pmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.1146 & -0.3854 & 0.5000 \\ 0.5000 & -0.4542 & -0.0458 \end{pmatrix} \cdot \begin{pmatrix} 255 \cdot R' \\ 255 \cdot G' \\ 255 \cdot B' \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \right) \cdot 2^{n-8}$$

for n-bit limited range

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \left(\begin{pmatrix} 0.1826 & 0.6142 & 0.0620 \\ -0.1007 & -0.3385 & 0.4392 \\ 0.4392 & -0.3990 & -0.0402 \end{pmatrix} \cdot \begin{pmatrix} 255 \cdot R' \\ 255 \cdot G' \\ 255 \cdot B' \end{pmatrix} + \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} \right) \cdot 2^{n-8}$$

- **4:2:0 chroma subsampling**

$$U_{xy} = \sum_{i=2 \cdot x}^{2 \cdot x+1} \sum_{j=2 \cdot y}^{2 \cdot y+1} \frac{C_{Bij}}{4}$$

$$V_{xy} = \sum_{i=2 \cdot x}^{2 \cdot x+1} \sum_{j=2 \cdot y}^{2 \cdot y+1} \frac{C_{Rij}}{4}$$

YUV formats

one cell represents one byte

(w, h) are image dimensions

$\mathsf{MSB}(z) = \left\lfloor \frac{z}{2^8} \right\rfloor$ (most significant byte)

$\mathsf{LSB}(z) = \left\{ \frac{z}{2^8} \right\} \cdot 2^8$ (least significant byte)

- **YV12 (8-bit planar 4:2:0)**

h × w

Y_{00}	Y_{10}	...
Y_{01}	Y_{11}	...
...	...	...

$\frac{h}{2}$ × $w/2$

V_{00}	V_{10}	...
V_{01}	V_{11}	...
...	...	...

$\frac{h}{2}$ × $w/2$

U_{00}	U_{10}	...
U_{01}	U_{11}	...
...	...	...

- **I420_10LE (10-bit planar 4:2:0)**

h × $2 \cdot w$

$\mathsf{LSB}(Y_{00})$	$\mathsf{MSB}(Y_{00})$	$\mathsf{LSB}(Y_{10})$	$\mathsf{MSB}(Y_{10})$	...
$\mathsf{LSB}(Y_{01})$	$\mathsf{MSB}(Y_{01})$	$\mathsf{LSB}(Y_{11})$	$\mathsf{MSB}(Y_{11})$	...
...	...	...	...	...

$$
\begin{array}{c}
\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{w} \\
\frac{h}{2}\left\{\begin{array}{|c|c|c|c|c|}
\hline
\mathsf{LSB}(U_{00}) & \mathsf{MSB}(U_{00}) & \mathsf{LSB}(U_{10}) & \mathsf{MSB}(U_{10}) & \dots \\
\hline
\mathsf{LSB}(U_{01}) & \mathsf{MSB}(U_{01}) & \mathsf{LSB}(U_{11}) & \mathsf{MSB}(U_{11}) & \dots \\
\hline
\dots & \dots & \dots & \dots & \dots \\
\hline
\end{array}\right.
\end{array}
$$

$$
\begin{array}{c}
\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{w} \\
\frac{h}{2}\left\{\begin{array}{|c|c|c|c|c|}
\hline
\mathsf{LSB}(V_{00}) & \mathsf{MSB}(V_{00}) & \mathsf{LSB}(V_{10}) & \mathsf{MSB}(V_{10}) & \dots \\
\hline
\mathsf{LSB}(V_{01}) & \mathsf{MSB}(V_{01}) & \mathsf{LSB}(V_{11}) & \mathsf{MSB}(V_{11}) & \dots \\
\hline
\dots & \dots & \dots & \dots & \dots \\
\hline
\end{array}\right.
\end{array}
$$

- **NV12 (8-bit semi-planar 4:2:0)**

$$
\begin{array}{c}
\overbrace{\qquad\qquad\qquad}^{w} \\
h\left\{\begin{array}{|c|c|c|}
\hline
Y_{00} & Y_{10} & \dots \\
\hline
Y_{01} & Y_{11} & \dots \\
\hline
\dots & \dots & \dots \\
\hline
\end{array}\right.
\end{array}
$$

$$
\begin{array}{c}
\overbrace{\qquad\qquad\qquad\qquad\qquad}^{w} \\
\frac{h}{2}\left\{\begin{array}{|c|c|c|c|c|}
\hline
U_{00} & V_{00} & U_{10} & V_{10} & \dots \\
\hline
U_{01} & V_{01} & U_{11} & V_{11} & \dots \\
\hline
\dots & \dots & \dots & \dots & \dots \\
\hline
\end{array}\right.
\end{array}
$$

- **P010 (10-bit semi-planar 4:2:0)**

$$
\begin{pmatrix} \hat{Y} \\ \hat{U} \\ \hat{V} \end{pmatrix} = \begin{pmatrix} Y \\ U \\ V \end{pmatrix} \cdot 2^6
$$

$$
\begin{array}{c}
\overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{2 \cdot w} \\
h\left\{\begin{array}{|c|c|c|c|c|}
\hline
\mathsf{LSB}(\hat{Y}_{00}) & \mathsf{MSB}(\hat{Y}_{00}) & \mathsf{LSB}(\hat{Y}_{10}) & \mathsf{MSB}(\hat{Y}_{10}) & \dots \\
\hline
\mathsf{LSB}(\hat{Y}_{01}) & \mathsf{MSB}(\hat{Y}_{01}) & \mathsf{LSB}(\hat{Y}_{11}) & \mathsf{MSB}(\hat{Y}_{11}) & \dots \\
\hline
\dots & \dots & \dots & \dots & \dots \\
\hline
\end{array}\right.
\end{array}
$$

$$
\begin{array}{c}
\overbrace{\qquad}^{2 \cdot w} \\
\frac{h}{2}\left\{\begin{array}{|c|c|c|c|c|c|c|c|c|c|}
\hline
\mathsf{LSB}(\hat{U}_{00}) & \mathsf{MSB}(\hat{U}_{00}) & \mathsf{LSB}(\hat{V}_{00}) & \mathsf{MSB}(\hat{V}_{00}) & \mathsf{LSB}(\hat{U}_{10}) & \mathsf{MSB}(\hat{U}_{10}) & \mathsf{LSB}(\hat{V}_{10}) & \mathsf{MSB}(\hat{V}_{10}) & \dots \\
\hline
\mathsf{LSB}(\hat{U}_{01}) & \mathsf{MSB}(\hat{U}_{01}) & \mathsf{LSB}(\hat{V}_{01}) & \mathsf{MSB}(\hat{V}_{01}) & \mathsf{LSB}(\hat{U}_{11}) & \mathsf{MSB}(\hat{U}_{11}) & \mathsf{LSB}(\hat{V}_{11}) & \mathsf{MSB}(\hat{V}_{11}) & \dots \\
\hline
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
\hline
\end{array}\right.
\end{array}
$$

parameters

- `output_format`:
  - `RGBA8` - 4 bytes per pixel 8-bit RGBA format with alpha channel set to `0xff`
  - `YV12_BT601` - BT.601 limited range YV12 format
  - `YV12_BT601_FR` - BT.601 full range YV12 format
  - `YV12_BT709` - BT.709 limited range YV12 format
  - `I420_10LE_BT601` - BT.601 limited range I420_10LE format
  - `I420_10LE_BT601_FR` - BT.601 full range I420_10LE format

- `I420_10LE_BT709` - BT.709 limited range I420_10LE format
- `NV12_BT601` - BT.601 limited range NV12 format
- `NV12_BT601_FR` - BT.601 full range NV12 format
- `NV12_BT709` - BT.709 limited range NV12 format
- `P010_BT601` - BT.601 limited range P010 format
- `P010_BT601_FR` - BT.601 full range P010 format
- `P010_BT709` - BT.709 limited range P010 format

`export_to_hostmem`

Copies data to CPU buffer converting to specified format. Supports same output formats as export_to_device component.

JSON configuration:

```
{
    "id": "export_to_hostmem",
    "type": "export_to_hostmem",
    "output_format": "NV12_BT601_FR",
    "output_name": "export_to_hostmem_out"
}
```

parameters

See parameters of `export_to_device` component.

`histogram`

Computes a histogram and exports it as an array of 32-bit integers to CPU buffer.

JSON configuration:

```
{
    "id": "hist",
    "type": "histogram",
    "output_format": "Histogram3Bayer256Int",
    "output_name": "histogram"
}
```

formula

$$whitepoint = bins - 1$$

$$out_{xy} = \sum_{(i,j)\in\Pi_y} \mathrm{I}_x(in_{ij})$$

$$x \in \{0, 1, 2, \ldots, whitepoint\}$$

$$y \in \mathrm{X}$$

$$\mathrm{I}_x(z) = \begin{cases} 0 & \frac{z}{white\_level} < \frac{x}{whitepoint} \\ 1 & \frac{x}{whitepoint} \le \frac{z}{white\_level} < \frac{x+1}{whitepoint} \\ 0 & \frac{x+1}{whitepoint} \le \frac{z}{white\_level} \end{cases} \qquad \text{(a)}$$

$\Pi_y = \{(i,j) \mid i,j \in \mathbb{N}_0, i < w, j < h, (i + c\_x) \bmod 2 + 2 \cdot ((j + c\_y) \bmod 2) \in \Upsilon_y\}$ (b)

where (w,h) are image dimensions, and (c_x, c_y) defines image Bayer pattern shift compared to RGGB

$\Upsilon_{\mathrm{V}} = \{0,1,2,3\}, \Upsilon_{\mathrm{R}} = \{0\}, \Upsilon_{\mathrm{G}} = \{1,2\}, \Upsilon_{\mathrm{G1}} = \{1\}, \Upsilon_{\mathrm{G2}} = \{2\}, \Upsilon_{\mathrm{B}} = \{3\}$

(a) defines whether value z falls into bin x. (b) defines pixel positions for specific color channel from X.

parameters

- `offset_x` (default 0)
- `offset_y` (default 0)
- `width` (optional)
- `height` (optional)
- `output_format` (<bins> is a power of 2):
  - `HistogramMono<bins>Int` - $\mathrm{X} = \{\mathrm{V}\}$
  - `Histogram3Bayer<bins>Int` - $\mathrm{X} = \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$
  - `Histogram4Bayer<bins>Int` - $\mathrm{X} = \{\mathrm{R}, \mathrm{G1}, \mathrm{G2}, \mathrm{B}\}$
  - `HistogramRGB256Int` - not yet documented
  - `HistogramParade256Int` - not yet documented

First 4 parameters define ROI for histogram computation, by default whole image is processed.

Image filters

`black_level`

Add-multiply filter, which subtracts black level (taken from image metadata) from each pixel and then scales the result, so that maximum (white level) stays the same.

JSON configuration:

```
{
    "id": "black_level",
    "type": "black_level"
}
```

formula

$$out = (in - black\_level) \cdot \frac{white\_level}{white\_level - black\_level}$$

`ffc`

Add-multiply filter, which subtracts dark frame from the image and corrects shading using flat field image.

JSON configuration:

```
{
    "id": "ffc",
    "type": "ffc",
    "flat_field": "flatfield-12.raw",
    "dark_field": "darkfield-12.raw",
    "bitdepth": 12
}
```

formula

$$D_{xy} = dark_{xy} \cdot 2^{in\_bitdepth - bitdepth}$$
$$F_{xy} = flat_{xy} \cdot 2^{in\_bitdepth - bitdepth}$$
$$G_{xy} = \frac{white\_level}{white\_level - \overline{D_{bayer}}} \cdot \frac{\overline{(F-D)_{bayer}}}{F_{xy} - D_{xy}}$$
$$out_{xy} = (in_{xy} - D_{xy}) \cdot \begin{cases} \frac{1}{8} & G_{xy} < \frac{1}{8} \\ G_{xy} & \frac{1}{8} \le G_{xy} \le 8 \\ 8 & G_{xy} > 8 \end{cases}$$

where $_{bayer}$ means such $\acute{x}$ and $\acute{y}$, that $\begin{cases} \acute{x} \bmod 2 = x \bmod 2 \\ \acute{y} \bmod 2 = y \bmod 2 \end{cases}$, or any $\acute{x}$ and $\acute{y}$ if image is monochrome

parameters

- `dark_field`: path to the file containing dark field image in raw 16-bit format
- `flat_field`: path to the file containing flat field image in raw 16-bit format
- `bitdepth` (optional): bit-depth of calibration files, by default the same as input bit-depth
- `width` (optional): width of calibration files, by default the same as input width
- `offset_x` (default 0)
- `offset_y` (default 0)

Last 2 parameters define position of the input image relative to calibration files. Last 3 parameters can be used to process cropped image without modifying the calibration files.

Note, that even if bit-depth is 8, calibration files still use 2-byte format with higher byte zeroed out.

`white_balance`
: Applies white balance to the image.

JSON configuration:

```
{
    "id": "wb",
    "type": "white_balance",
    "algorithm": "simple",
    "comp_min": 0.0,
    "comp_max": 1.0
}
```

formula for simple algorithm

$out_{xy} = in_{xy} \cdot gain_{\Pi(x,y)}$

where $gain_i$ is white balance settings for input image

$i \in \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$ or $i \in \{\mathrm{R}, \mathrm{G1}, \mathrm{G2}, \mathrm{B}\}$ depending on which white balance settings are provided

$\Pi(x, y) = \Upsilon\Big((x \bmod 2 + 2 \cdot (y \bmod 2) + c) \bmod 4\Big)$

where c defines image Bayer pattern shift compared to RGGB

$\Upsilon(0) = \mathrm{R}, \Upsilon(1) = \mathrm{G}$ or $\mathrm{G1}, \Upsilon(2) = \mathrm{G}$ or $\mathrm{G2}, \Upsilon(3) = \mathrm{B}$

formula for histogram stretch algorithm

$$cut_i = off_i + \frac{comp\_max - comp\_min}{gain_i}$$

where off_i and $gain_i$ are white balance settings for input image

$i \in \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$

$$out_{xy} = (2^{16} - 1) \cdot \begin{cases} comp\_min \cdot \frac{in_{xy}}{white\_level \cdot off_{\Pi(x,y)}} & \frac{in_{xy}}{white\_level} < off_{\Pi(x,y)} \\ comp\_min + gain_{\Pi(x,y)} \cdot (\frac{in_{xy}}{white\_level} - off_{\Pi(x,y)}) & off_{\Pi(x,y)} \leq \frac{in_{xy}}{white\_level} \leq cut_{\Pi(x,y)} \\ comp\_max + \frac{1 - comp\_max}{1 - cut_{\Pi(x,y)}} \cdot (\frac{in_{xy}}{white\_level} - cut_{\Pi(x,y)}) & cut_{\Pi(x,y)} < \frac{in_{xy}}{white\_level} \end{cases}$$

$\Pi(x, y) = \Upsilon\Big((x \bmod 2 + 2 \cdot (y \bmod 2) + c) \bmod 4\Big)$

where c defines image Bayer pattern shift compared to RGGB

$\Upsilon(0) = \mathrm{R}, \Upsilon(1) = \mathrm{G}, \Upsilon(2) = \mathrm{G}, \Upsilon(3) = \mathrm{B}$

parameters

- `algorithm` (default ”simple”)
  - `simple` - per-channel multiplication by gain value, doesn’t change bit-depth
  - `stretch` - histogram stretch implemented using LUT with 16-bit output (for 16-bit input 14-bit LUT is used together with linear interpolation)
- `comp_min` (default 0.0) - maximum normalized value for shadow compression section
- `comp_max` (default 1.0) - minimum normalized value for highlights compression section

Last 2 parameters define values of corresponding variables in histogram stretch formula, and thus have an effect only when `algorithm` is set to `stretch`.

With default settings histogram stretch algorithm is equivalent to a combination of per-channel black level (offset) and simple white balance (gain).

`color_correction`

Transforms image colors by matrix multiplying RGB color values of each pixel by specified 3x3 **color correction matrix**.

JSON configuration:

```
{
    "id": "color_correction",
    "type": "color_correction",
    "matrix": [ 1.0, 0.0, 0.0,
                0.0, 1.0, 0.0,
                0.0, 0.0, 1.0 ]
}
```

formula

$$\begin{pmatrix} R_{out} \\ G_{out} \\ B_{out} \end{pmatrix} = \begin{pmatrix} M_{00} & M_{01} & M_{02} \\ M_{10} & M_{11} & M_{12} \\ M_{20} & M_{21} & M_{22} \end{pmatrix} \cdot \begin{pmatrix} R_{in} \\ G_{in} \\ B_{in} \end{pmatrix}$$

parameters

- `matrix`: color correction matrix M in row scan order

`demosaic`
Transforms raw Bayer image into RGB image.

JSON configuration:

```
{
    "id": "demosaic",
    "type": "demosaic",
    "algorithm": "HQLI"
}
```

parameters

- `algorithm`:
  - `HQLI` - High Quality Linear Interpolation, window 5×5, avg. PSNR ~36 dB for Kodak data set
  - `L7` - High Quality Linear Interpolation, window 7×7, avg. PSNR ~37.1 dB (SSIM ~0.971) for Kodak data set, doesn't support 8-bit input
  - `DFPD` - Directional Filtering and a Posteriori Decision, window 11×11, avg. PSNR ~39 dB for Kodak data set
  - `MG` - Multiple Gradients, window 23×23, avg. PSNR ~40.5 dB for Kodak data set, doesn't support 8-bit input

---

`crop`
Crops the image.

JSON configuration:

```
{
    "id": "crop",
    "type": "crop",
    "offset_x": 0,
    "offset_y": 0,
    "out_width": 4096,
    "out_height": 4096
}
```

parameters

- `out_width`
- `out_height`
- `offset_x`
- `offset_y`

These parameters defines crop area.

`resizer`
Scales the image using Lanczos algorithm. Aspect ratio might not be preserved.

JSON configuration:

```
{
    "id": "resizer",
    "type": "resizer",
    "out_width": 512,
    "out_height": 376
}
```

parameters

- `out_width`
- `out_height`

These parameters defines dimensions of the output image.

`gamma8`, `gamma12`, `gamma16`
Applies gamma curve using LUT with 8-bit, 12-bit or 16-bit output. For 16-bit input 14-bit LUT is used together with linear interpolation.

JSON configuration:

```
{
    "id": "gamma8",
    "type": "gamma8",
    "function": "gamma",
    "linear": 0.0,
    "power": 1.0
}
```

formula

$$out = (2^{out\_bitdepth} - 1) \cdot \Gamma\left(\frac{in}{white\_level}\right)$$

- **BT.709-like gamma**

$$\Gamma(x) = \begin{cases} c \cdot x & x < linear \\ a \cdot x^{power} - b & x \geq linear \end{cases}$$

where a, b and c are calculated, so that $\Gamma(x)$ is smooth and passes through (0, 0) and (1, 1)

- **Hybrid Log-Gamma**

$$\Gamma(x) = \begin{cases} \sqrt{3 \cdot x} & x \leq \frac{1}{12} \\ a \cdot \ln(12 \cdot x - b) + c & x > \frac{1}{12} \end{cases}$$

$a = 0.17883277$

$b = 0.28466892$

$c = 0.55991073$

parameters

- `function`:
  - `gamma` (default) - use BT.709-like gamma function
  - `hlg` - use Hybrid Log-Gamma function
- `power` (default 1.0)
- `linear` (default 0.0)

Last 2 parameters define values of corresponding variables in BT.709-like gamma formula, and thus have an effect only when `function` is set to `gamma`.

`bitdepth`

Changes bit-depth of the image using zero-filling shift operation.

JSON configuration:

```
{
    "id": "bitdepth",
    "type": "bitdepth",
    "bitdepth": 8
}
```

formula

$$out = in \cdot 2^{bitdepth - in\_bitdepth}$$

parameters

- `bitdepth` (optional): output bit-depth, by default converts 10-bit format and 14-bit format to 16-bit leaving others as-is

5 IFF SDK library interface

IFF SDK provides the C library interface for managing image processing chains within the IFF control flow. The interface of SDK library is defined by **`iff.h`** header file in the IFF SDK package.

5.1 Functions

5.1.1 `iff_initialize()`

```
void iff_initialize(const char* config);
```

Initialize new instance of IFF framework or increment its usage count if it has already been initialized by the calling process. Should be called before any other SDK library function call. For each call of this function process must do a corresponding call of `iff_finalize()` function. If an instance of IFF framework is already initialized, parameter `config` is ignored.

Parameters:

`config`	Configuration of IFF framework in JSON format

5.1.2 `iff_finalize()`

```
void iff_finalize();
```

Decrement usage count of IFF framework instance by calling process. When usage count reaches zero, instance is released and all processing chains within this instance are destroyed.

5.1.3 `iff_log()`

```
void iff_log(const char* level, const char* message);
```

Adds a message to IFF SDK log, unless currently configured log level is greater than specified message severity.

Parameters:

`level`	Message severity, one of the following constants: `IFF_LOG_LEVEL_DEBUG`, `IFF_LOG_LEVEL_WARNING`, `IFF_LOG_LEVEL_ERROR`, `IFF_LOG_LEVEL_INFO` (always logged)
`message`	Message to be logged

5.1.4 iff_create_chain()

```
iff_chain_handle_t iff_create_chain(const char* chain_config, iff_error_handler_t ↩
    on_error);
```

Create a new IFF processing chain according to passed configuration.

Parameters:

chain_config	Configuration of IFF chain to create in JSON format. See Chain description format
on_error	Pointer to a function that is called if error occurred during processing chain lifetime. See iff_error_handler_t

---

Returns:
Handle of newly created chain

5.1.5 iff_release_chain()

```
void iff_release_chain(iff_chain_handle_t chain_handle);
```

Finalize processing chain and release all its resources.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function

---

5.1.6 iff_get_params()

```
void iff_get_params(iff_chain_handle_t chain_handle, const char* params, ↩
    iff_result_handler_t ret_func);
```

Get values of given chain elements parameters. Can request parameters from multiple elements at once.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function
params	Elements parameters names to get in JSON format. See Get parameters input format
ret_func	Pointer to a function that is called by SDK to return values of requested elements parameters. See iff_result_handler_t

---

5.1.7 iff_set_params()

```
void iff_set_params(iff_chain_handle_t chain_handle, const char* params);
```

Set chain elements parameters. Can set parameters for multiple chain elements at once.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function
params	Chain elements parameters and its values to set. See Set parameters input format

---

5.1.8 iff_execute()

```
void iff_execute(iff_chain_handle_t chain_handle, const char* command);
```

Request execution of the specified command from the chain element.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function
command	Command to execute and its parameters if any in JSON format. See Execute input format

---

5.1.9 iff_set_callback()

```
void iff_set_callback(iff_chain_handle_t chain_handle, const char* name,  ↩
   iff_callback_t callback);
```

Set the given function to the specified element callback.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function
name	Element callback name in the format **<element ID>/<callback name>**
callback	Pointer to callback function. See iff_callback_t

5.1.10 iff_set_export_callback()

```
void iff_set_export_callback(iff_chain_handle_t chain_handle, const char*  ↩
   exporter_id, iff_frame_export_function_t export_func, void* private_data);
```

Set the given function to the specified exporter element (see frame_exporter) as export callback, in which a pointer to the frame data will be passed from IFF SDK library to the user code.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function
exporter_id	ID of the exporter element. See frame_exporter
export_func	Pointer to export callback function. See iff_frame_export_function_t
private_data	Pointer to the user data. This pointer will be passed as parameter to export_func function with each invocation

5.2 Structures

5.2.1 iff_image_metadata

Image metadata structure contains parameters of a specific processed image.

Structure definition:

```
typedef struct iff_wb_params
{
    float r;
    float g1;
    float g2;
    float b;
    float r_off;
    float g1_off;
    float g2_off;
    float b_off;
} iff_wb_params;

typedef struct iff_image_metadata
{
    size_t   padding;

    uint32_t width;
    uint32_t height;
    uint32_t offset_x;
    uint32_t offset_y;

    uint64_t ts;
    uint64_t ntp_time;

    uint32_t black_level;
    unsigned int exposure;
    float gain;

    iff_wb_params wb;
```

```
    unsigned char sequence_id;
} iff_image_metadata;
```

Table 1: **Members:**

`padding`	Image padding in bytes.
`width`	Image width in pixels.
`height`	Image height in pixels.
`offset_x`	X Offset of ROI or Crop position.
`offset_y`	Y Offset of ROI or Crop position.
`ts`	Image timestamp provided by source.
`ntp_time`	Image timestamp in NTP format. See Specification.
`black_level`	Image black level.
`exposure`	Image exposure time in microseconds (used only if image source is a camera).
`gain`	Image gain in dB (used only if image source is a camera).
`wb`	Image white balance coefficients.
`sequence_id`	ID of a dispatch session within which given image was dispatched provided by source.

5.3 Types

5.3.1 iff_error_handler_t

```
typedef void(*iff_error_handler_t)(const char* element_name, int error_code);
```

Function pointer of this type must be passed to `iff_create_chain()` function when creating a new chain. IFF will call the function at the given pointer whenever an error occurs while chain is processing the image or executing a user request.

Parameters:

`element_name`	ID of the chain element that triggered the error
`error_code`	Code of an error

5.3.2 iff_result_handler_t

```
typedef void(*iff_result_handler_t)(const char* params);
```

A function pointer of this type should be passed as a parameter to the `iff_get_params()` call. IFF will call the function at the given pointer to return a JSON string containing the values of the requested parameters. This JSON string will be passed to the function as a parameter.

Parameters:

`params`	Values of requested chain elements parameters in JSON format

---

5.3.3 iff_callback_t

```
typedef void(*iff_callback_t)(const char* callback_data);
```

Function pointer of this type must be passed to `iff_set_callback()` function call. This function will be set to element callback with specified name.

Parameters:

`callback_data`	Data returned by element callback in JSON format

5.3.4 iff_frame_export_function_t

```
typedef void(*iff_frame_export_function_t)(const void* data, size_t size,  ↩
    iff_image_metadata* metadata, void* private_data);
```

Function pointer of this type must be passed to `iff_set_export_callback()` function call. The function at the given pointer is called by exporter element when a new frame is received to send it to the client code across IFF SDK library boundaries. After this function returns, the image is released by API and is no longer valid.

Parameters:

`data`	Pointer to image data. Could be both GPU or CPU memory pointer. After export function returns, this pointer is released by IFF SDK and is no longer valid.
`size`	Size of image data in bytes
`metadata`	Pointer to the image metadata structure. See `iff_image_metadata`
`private_data`	Pointer to the user data that was passed to `iff_set_export_callback()` call

6 IFF SDK configuration

When writing application using IFF SDK, as the first step you always need to initialize SDK framework.

6.1 Initializing IFF

Before the IFF SDK can be used, `iff_initialize()` has to be called from the application process. This call will perform the necessary initialization of IFF context according to provided framework configuration in JSON format.

6.2 Framework configuration format

framework configuration example:

```
{
    "logfile": "",
    "log_level": "WARNING",
    "set_terminate": false,

    "service_threads": 0,

    "enable_control_interface": false,
    "control_interface_base_url": "/chains",

    "devices": [
        {
            "id": "cpu_dev",
            "type": "cpu"
        },
        {
            "id": "cuda_dev",
            "type": "cuda",
            "device_number": 0
        }
    ],

    "services": {
        "rtsp_server": {
            "host": "192.168.55.1",
            "port": 8554,
            "mtu": 1500,
            "listen_depth": 9,
            "read_buffer_size": 16384,
            "receive_buffer_size": 4194304,
            "session_timeout": 60
        },
        "http_server": {
            "host": "0.0.0.0",
            "port": 8080,
            "listen_depth": 9
        }
    }
}
```

common settings

- `logfile` (default "") - log file path, if empty IFF will output log information to stdout
- `log_level` (default "WARNING") - minimal level of messages to report into log file, one of the following values (in the ascending order of severity):
  - `DEBUG`
  - `WARNING`
  - `ERROR`
  - `FATAL`
- `set_terminate` (default false) - whether to set terminate handler that logs unhandled C++ exceptions
- `service_threads` (default 0) - number of threads in the main framework service pool, if set to zero number of CPU cores is used
- `enable_control_interface` (default false) - whether to enable HTTP control interface for each created chain
- `control_interface_base_url` (default "/chains") - base relative URL for chain control interface within HTTP server (control interface URL for each chain will be `<control_interface_base_url ID>`)

`devices`

This section describes the devices used by the framework (i.e. GPU and CPU).

device parameters

- `id` - device ID
- `type` - type of the device (`cpu` or `cuda`)
- `device_number` (default 0) - sequence number of the device (used only for CUDA devices)

`services/rtsp_server`

RTSP server configuration.

parameters

- `host` - server IP address (can't be 0.0.0.0)
- `port` (default 8554) - server port
- `MTU` (default 1500) - network MTU
- `listen_depth` (default 9) - depth of the listen queue
- `read_buffer_size` (default 16384) - buffer size when reading from an UDP socket
- `receive_buffer_size` (default 4194304) - OS receive buffer size of an UDP socket
- `session_timeout` (default 60) - keep-alive timeout for a session

`services/http_server`

HTTP server for chain control interface configuration.

parameters

- `host` (default "0.0.0.0") - server IP address (can be 0.0.0.0 to listen on all addresses)
- `port` (default 8080) - server port
- `listen_depth` (default 9) - depth of the listen queue

6.3 Chain description format

IFF creates processing chains based on their description in JSON format. Since the processing chain is an directed acyclic graph, its description is a set of vertices (Elements) interconnected by edges (Connections). Thus, in order to define any processing chain, a list of elements and a list of connections between their inputs and outputs are necessary. In addition, IFF allows, if necessary, to define a list of external parameter control for each element of the chain.

Chain definition example:

```
{
    "id": "main",

    "elements": [
        {
            "id": "cam",
            "type": "xicamera",
            "cpu_device_id": "cpu_dev",
            "serial_number": "XECAS1930002",
            "image_format": "RAW8",
            "custom_params": [
                { "bpc":                              1 },
                { "column_fpn_correction":            1 },
                { "row_fpn_correction":               1 },
                { "column_black_offset_correction": 1 },
                { "row_black_offset_correction":     1 }
            ],
            "exposure": 10000,
            "fps": 30.0,
            "gain": 0.0
        },
        {
            "id": "writer",
            "type": "dng_writer",
            "cpu_device_id": "cpu_dev",
            "filename_template": "{utc_time}.dng"
        },
        {
            "id": "gpuproc",
            "type": "cuda_processor",
            "cpu_device_id": "cpu_dev",
            "gpu_device_id": "cuda_dev",
            "elements": [
                { "id": "import_from_host", "type": "import_from_host" },
                { "id": "black_level",      "type": "black_level" },
                { "id": "white_balance",    "type": "white_balance" },
                { "id": "demosaic",         "type": "demosaic",         "algorithm
                    ": "HQLI" },
                { "id": "color_correction", "type": "color_correction", "matrix":
                    [ 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0 ] },
                { "id": "gamma",            "type": "gamma8",           "linear":
                    0.018, "power": 0.45 },
                { "id": "export_to_device", "type": "export_to_device", "
                    output_format": "NV12_BT709",           "output_name": "yuv"
                    },
                { "id": "hist",             "type": "histogram",        "
                    output_format": "Histogram4Bayer256Int", "output_name": "
                    histogram" }
```

```
            ],
            "connections": [
                { "src": "import_from_host", "dst": "black_level" },
                { "src": "black_level",      "dst": "white_balance" },
                { "src": "white_balance",    "dst": "demosaic" },
                { "src": "demosaic",         "dst": "color_correction" },
                { "src": "color_correction", "dst": "gamma" },
                { "src": "gamma",            "dst": "export_to_device" },
                { "src": "black_level",      "dst": "hist" }
            ]
        },
        {
            "id": "autoctrl",
            "type": "awb_aec",
            "cpu_device_id": "cpu_dev",
            "autostart": true,
            "aec_enabled": true,
            "awb_enabled": true,
            "max_exposure": 33000
        },
        {
            "id": "nvenc",
            "type": "encoder",
            "encoder_type": "nvidia",
            "cpu_device_id": "cpu_dev",
            "gpu_device_id": "cuda_dev",
            "max_processing_count": 3,
            "codec": "H264",
            "bitrate": 10000000,
            "fps": 30.0,
            "max_performance": true
        },
        {
            "id": "mon",
            "type": "sub_monitor"
        },
        {
            "id": "netstream",
            "type": "rtsp_stream",
            "relative_uri": "/cam"
        }
    ],
    "connections": [
        { "src": "cam",                          "dst": "writer" },
        { "src": "cam",                          "dst": "gpuproc" },
        { "src": "gpuproc->histogram",           "dst": "autoctrl", "type": "weak
          " },
        { "src": "gpuproc->yuv",                 "dst": "nvenc" },
        { "src": "nvenc",                        "dst": "mon" },
        { "src": "mon",                          "dst": "netstream" }
    ],
    "parametercontrol": [
        { "origin": "autoctrl/wb_callback",       "target": "cam" },
        { "origin": "autoctrl/exposure_callback", "target": "cam" }
    ],
    "commandcalls": [
        { "origin": "mon/on_new_consumer",        "target": "nvenc", "execute": {
```

```
            "command": "force_idr" } }
    ]
}
```

Important
Each chain created by the same IFF SDK instance must have a unique `id`

6.3.1 Elements

The `elements` section of the chain description contains the configuration of the elements that make up the chain. For more information about chain elements configuration see IFF components.

6.3.2 Connections

The `connections` section of the chain description defines how elements described above are linked together into the chain. There are two types of connections between chain elements: `weak` and `strong`. Weakly connected elements do not trigger their sources to start dispatching, but they do receive frames if their source has strongly connected consumers.

Each connection has the following attributes:

attributes

- `src` - ID and output name of given connection source element (element dispatching images) in one of the following formats:
  - `<src element id>` (for example `nvenc`) when referring to element's default output (usually when it has only one output)
  - `<src element id>-><output name>` (for example `gpuproc->nv12`) otherwise
- `dst` - ID and input name of given connection destination element (element receiving images) in one of the following formats:
  - `<dst element id>` (for example `nvenc`) when referring to element's default input (usually when it has only one input)
  - `<dst element id>-><input name>` (for example `nvenc->in`) otherwise
- `type` (default "strong") - type of the given connection, one of the following values:
  - `strong`
  - `weak`

6.3.3 Parameter control list

The `parametercontrol` section of the chain description defines parameters control links between the elements. Parameters control links are useful when one element needs to set some parameters to another. For example in auto white balance implementation `awb_aec` component should set white balance coefficients in its `wb_callback` to the camera component.

Each connection has the following attributes:

attributes

- `origin` - ID and callback name of controlling element
- `target` - ID of controlled element

6.3.4 Command call list

The `commandcalls` section of the chain description defines command callback links between the elements. Command callback links are useful when one element needs to request command execution from another element. For example in RTSP streaming implementation `sub_monitor` component should execute `force_idr` `encoder` element command in its `on_new_consumer` callback.

Each connection has the following attributes:

attributes

- `origin` - ID and callback name of controlling element
- `target` - ID of controlled element
- `execute` - command description in execute input format without the element ID

6.4 Input formats of `controllable` interface functions

IFF chains and components inherit `controllable` interface through element. This interface allows to get and set parameters to chain components and to send commands to them. Access to this functionality in the SDK library interface is given by functions `iff_get_params()`, `iff_set_params()` and `iff_execute()`.

6.4.1 Get parameters input format

iff_get_params() input example:

```
{
    "camera1": {
        "params": [
            "exposure",
            "gain",
            "wb"
        ]
    },
    "encoder1": {
        "params": [
            "codec",
            "fps",
            "bitrate"
        ]
    }
}
```

Input parameter of `iff_get_params()` function is a JSON string of the format shown above. IFF allows to get parameters of multiple elements at once with one request. To get parameters of the needed chain elements, it needs to specify their IDs as first-level keys. The `params` array contains a list of the required parameters names of the corresponding element.

6.4.2 Set parameters input format

iff_set_params() input example:

```
{
    "camera1": {
        "exposure": 15,
        "gain": 0.0,
        "wb": {
            "r": 1.0,
            "g": 1.0,
            "b": 1.0
        }
    },
    "cudaproc1": {
        "crop_positions": {
            "offset_x": 400,
            "offset_y": 300
        }
    }
}
```

First level keys are the IDs of elements that need to be set parameters. The element parameters have the same format as in the chain description that is passed to `iff_create_chain()` function.

For a list of supported parameters for a particular element, see IFF components.

6.4.3 Execute input format

iff_execute() input example:

```
{
    "writer1": {
        "command": "on",
        "args": {
            "filename": "test.h265"
        }
    }
}
```

As input `iff_execute()` accepts a JSON string where key is ID of the chain element you want to send command to. `command` is a name of the command to be executed by this element. `args` contains names and corresponding values of the command options.

6.5 Chain control via HTTP

IFF processing chains can be controlled via HTTP interface. To enable this interface set `enable_control_interface` option to `true`. For HTTP server configuration and other control interface options see Framework configuration format.

URL of control interface for each chain depends on value of `control_interface_base_url` option. For each chain three control URLs are created:

```
http://<HTTP_SERVER_HOST>:<HTTP_SERVER_PORT>/chains/<chain ID>/get_params
http://<HTTP_SERVER_HOST>:<HTTP_SERVER_PORT>/chains/<chain ID>/set_params
http://<HTTP_SERVER_HOST>:<HTTP_SERVER_PORT>/chains/<chain ID>/execute
```

Each of these URLs allows you to send the corresponding command to the chain:

commands

- `get_params` - HTTP POST JSON to this URL calls `iff_get_params()` function of the corresponding chain (for JSON input format see Get parameters input format)
- `set_params` - HTTP POST JSON to this URL calls `iff_set_params()` function of the corresponding chain (for JSON input format see Set parameters input format)
- `execute` - HTTP POST JSON to this URL calls `iff_execute()` function of the corresponding chain (for JSON input format see Execute input format)

For more details about chains control functionality see Input formats of `controllable` interface functions section.

6.5.1 Curl command examples

get_params example:

```
curl -d '{ "cam": { "params": [ "exposure", "gain", "wb" ] }, "nvenc": { "params": ↩
    [ "codec", "fps", "bitrate" ] } }' -X POST http://127.0.0.1:8080/chains/main/ ↩
   get_params
```

This example shows how to get `exposure`, `gain` and `wb` parameters of element with ID `cam` and `codec`, `fps` and `bitrate` parameters of element with ID `nvenc` of chain `chain1`.

set_params example:

```
curl -d '{ "cam": { "exposure": 15000, "gain": 2.0 } }' -X POST http ↩
   ://127.0.0.1:8080/chains/main/set_params
```

This example shows how to set the camera's `cam` `exposure` and `gain` parameters.

execute example:

```
curl -d '{ "writer": { "command": "on", "args": { "frames_count": 1 } } }' -X POST ↩
    http://127.0.0.1:8080/chains/main/execute
```

This example shows how to send command `on` with runtime parameter `filename` to `writer` element of chain `chain1`.

7 Sample applications

7.1 `farsight`

Most basic and general sample application is called `farsight` and is located in `samples/01_streaming` directory of IFF SDK package. It comes with example configuration file (`farsight.json`) demonstrating the following functionality:

- acquisition from XIMEA camera
- writing of raw data to DNG files
- color pre-processing on GPU:
  - black level subtraction
  - histogram calculation
  - white balance
  - demosaicing
  - color correction
  - gamma
  - image format conversion
- automatic control of exposure time and white balance
- H.264 encoding
- RTSP streaming
- HTTP control interface

7.2 `imagebroker`

`imagebroker` application demonstrates how to export images to the user code across IFF SDK library boundaries. Application is located in `samples/02_export` directory of IFF SDK package. It comes with example configuration file (`imagebroker.json`) providing the following functionality:

- acquisition from XIMEA camera
- color pre-processing on GPU:
  - black level subtraction
  - histogram calculation
  - white balance
  - demosaicing
  - color correction (swapping R and B channels in given application)
  - gamma
  - image format conversion
- automatic control of white balance
- export image to the client code

Additionally example code renders images on the screen using OpenCV library, which should be installed in the system (minimal required version is 4.5.2).

7.3 crowsnest

Web interface sample called crowsnest demonstrates the possibility to control runtime parameters of IFF SDK pipeline and preview the video stream through an ordinary web browser. It is located in samples/03_webrtc directory of IFF SDK package. Web application code is based on Vue.js framework. Janus server is used to convert RTSP stream (as provided by IFF SDK) to WebRTC protocol supported by modern web browsers. nginx server is a standard solution to serve the web interface and proxy connections to IFF SDK and Janus control interface. farsight sample application can be used to run a compatible IFF SDK pipeline. User interface is self-documented in ”About” tab of the presented web page.

7.3.1 Installation

linux/install.sh installation script is provided as a reference. It was tested on Ubuntu 20.04, NVIDIA Jetson Linux (L4T) 32.7 and 35.4. On success it prints out instructions for final setup steps.

7.3.2 Deployment of modifications

The following commands should be used to deploy changes made to web interface source code (assuming default installation configuration as described above):

```
export PATH=/opt/mrtech/bin:"$PATH"
npm run build
cp -RT dist/ /opt/mrtech/var/www/html/
```

A Changelog

A.1 Version 1.7

- Added `v4l2cam` component.
- Migrated to new NVENC presets in `encoder` component to ensure compatibility with future releases of NVIDIA GPU drivers. Support for old presets is to be removed by NVIDIA in 2024 starting with driver version R550. `config_preset` and `rc_mode` parameters may have to be adjusted (and new `preset_tuning` and `multipass` parameters set) according to NVENC Preset Migration Guide.
- Various bug fixes.

A.2 Version 1.6

- Expanded NVIDIA Jetson Linux (L4T) support up to version 35, bringing capability to run on NVIDIA Jetson Orin modules.
- Fixed detection of newly connected cameras in `xicamera` source component.

A.3 Version 1.5

- Added `crowsnest` web interface sample.
- Added `metadata_exporter` component.

A.4 Version 1.4

- Added `genicam` component.
- Added support for 12-bit packed input formats to `cuda_processor`.
- Expanded NVIDIA GPU support up to Ada Lovelace architecture (compute capability 8.x). GPU driver update may be required after upgrading to this IFF SDK version.
- Added set_terminate parameter to framework configuration format.
- Fixed documentation of trigger-related features.
- Various bug fixes and minor improvements.

A.5 Version 1.3

- Added logging function to the C library interface.
- Enhanced auto white balance algorithm to better handle under- and over-exposure.
- Fixed writing of non-square TIFF/DNG files in dng_writer.
- Fixed compatibility of RTSP stream with WebRTC standard.
- `bitrate` parameter of encoder component can now be modified at runtime.
- Added `repeat_spspps`, `profile` and `level` parameters to encoder component.
- Added `force_idr` command to encoder component.

- Added sub_monitor component.
- Added commandcalls section to the chain description format.
- Added `session_timeout` parameter to rtsp_server settings.
- Other minor enhancements and bug fixes.

A.6 Version 1.2

- Added Chain control via HTTP.
- Incompatible change: Framework configuration format used for `iff_initialize()` call is now a value (JSON object) of what previously was `iff` top-level key.

A.7 Version 1.1

No functional changes, only documentation update.

A.8 Version 1.0

Initial release.

B License notices

JSON for Modern C++

IFF SDK and sample applications use JSON for Modern C++ library under the MIT License.

Copyright © 2013-2022 Niels Lohmann

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The class contains the UTF-8 Decoder from Bjoern Hoehrmann which is licensed under the MIT License (see above). Copyright © 2008-2009 Björn Hoehrmann <bjoern@hoehrmann.de>

The class contains a slightly modified version of the Grisu2 algorithm from Florian Loitsch which is licensed under the MIT License (see above). Copyright © 2009 Florian Loitsch

The class contains a copy of Hedley from Evan Nemerson which is licensed as CC0-1.0.

GenICam

GenICam edition of IFF SDK uses GenICam libraries under GenICam license.

Copyright (c) EMVA and contributors (see source files)

All rights reserved

Redistribution and use in source and binary forms, without modification, are permitted provided that the following conditions are met:

~ Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

~ Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

~ Neither the name of the GenICam standard group nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ”AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.