MRTech IFF SDK technical manual

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 2023	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
1.8	July 2024	IFF SDK release v1.8	MS
1.8.1	July 2024	IFF SDK release v1.8.1	MS
1.9	December 2024	IFF SDK release v1.9	MS
2.0	April 2025	IFF SDK release v2.0	MS
2.0.1	April 2025	IFF SDK release v2.0.1	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.

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

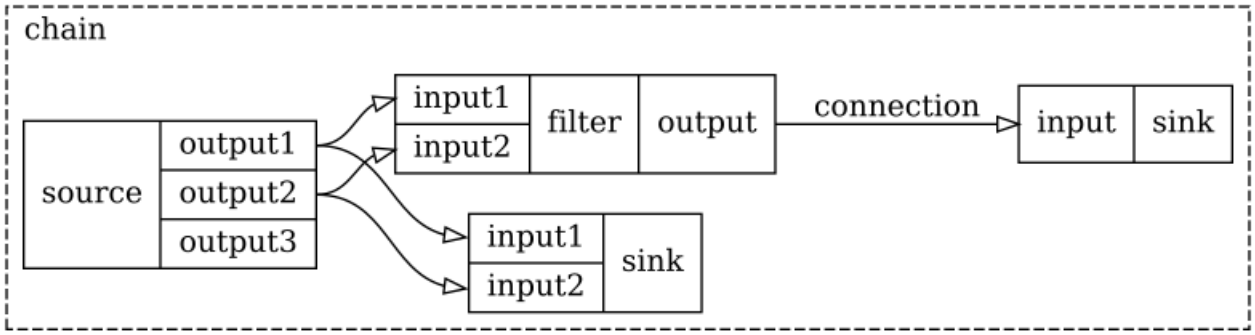

Figure 1: Pipeline

- 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
   b. neoAPI from Baumer
   c. Arena SDK from LUCID
   d. GenTL Producer from The Imaging Source
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 and C++ header files
   b. `lib` - shared libraries
   c. `licenses_3rdparty` - license texts of third party software used by IFF SDK
   d. `python` - Python bindings source code
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 C/C++ sample:
   - on Linux:

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

   - in Windows developer shell:

```
cd samples/01_streaming
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -G Ninja ..
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.

3.3.1 Python

IFF SDK package comes with pre-built Python bindings library (located in `sdk/lib` directory) for the following Python versions:

- CPython 3.8 on Linux;
- CPython 3.12 on Windows.

In order to use IFF SDK with another Python version rebuild the bindings with the following commands:

- on Linux:

```
cd sdk/python
mkdir build
cd build
cmake ..
cmake --build .
cmake --install .
```

- in Windows developer shell:

```
cd sdk/python
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -G Ninja ..
cmake --build .
cmake --install .
```

To use IFF SDK Python bindings there are two options:

1. copy the contents of `sdk/lib` directory next to the Python script (e.g. into `samples/01_streaming_py` directory);
2. Linux-only: set `PYTHONPATH` environment variable to the path to `sdk/lib` directory.

Note
For XIMEA edition on Windows it may be required to copy `C:/XIMEA/API/xiAPI/xiapi64.dll` file next to the Python script.

For example, to run `farsightpy` sample on Linux execute the following commands:

```
cd samples/01_streaming_py
export PYTHONPATH="$PWD/../../sdk/lib"
python3 ./farsight.py
```

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` (default) - 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` (default) - 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 commands:

- `start` - makes source start dispatch images
  execution result format:

```
{ "success": true }
```

- `stop` - makes source stop dispatch images
  execution result format:

```
{ "success": true }
```

- `trigger` - makes source dispatch an image, if it's in software trigger mode
  execution result format:

```
{ "success": true }
```

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,
    "max_buffers_queue_size": 2,
    "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 }
    ],
    "black_level": 0,
```

```
    "exposure": 10000,
    "gain": 0.0,
    "fps": 0.0,
    "auto_white_balance": false,
    "wb": {
        "dcp_file": "color_profile.dcp",
        "temperature": 5003,
        "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), for example:
  - Basler:
    - Ethernet cameras:
      - `/opt/pylon/lib/gentlproducer/gtl/ProducerGEV.cti` - Linux
      - `C:/Program Files/Basler/pylon 7/Runtime/x64/ProducerGEV.cti` - Windows
    - USB cameras:
      - `/opt/pylon/lib/gentlproducer/gtl/ProducerU3V.cti` - Linux
      - `C:/Program Files/Basler/pylon 7/Runtime/x64/ProducerU3V.cti` - Windows
  - Baumer (replace `/path/to/` with path to the directory with neoAPI C++ package contents):
    - Ethernet cameras:
      - `/path/to/lib/libbgapi2_gige.cti` - Linux
      - `/path/to/bin/bgapi2_gige.cti` - Windows
    - USB cameras:
      - `/path/to/lib/libbgapi2_usb.cti` - Linux
      - `/path/to/bin/bgapi2_usb.cti` - Windows
  - LUCID (on Linux replace `/path/to/` with path to the directory, where Arena SDK package was unpacked):
    - `/path/to/ArenaSDK_Linux_ARM64/lib/GenTL_LUCID.cti` - Linux ARM64
    - `/path/to/ArenaSDK_Linux_x64/lib64/GenTL_LUCID.cti` - Linux x86_64
    - `C:/Program Files/Lucid Vision Labs/Arena SDK/x64Release/GenTL_LUCID_v140.cti` - Windows
  - The Imaging Source:
    - Ethernet cameras:
      - `/usr/lib/aarch64-linux-gnu/gentl/tis/libic4-gentl-gev.cti` - Linux ARM64
      - `/usr/lib/x86_64-linux-gnu/gentl/tis/libic4-gentl-gev.cti` - Linux x86_64
      - `C:/Program Files/The Imaging Source Europe GmbH/IC4 GenTL Driver for GigEVision Devices/bin/ic4-gentl-gev.cti` - Windows
    - USB cameras:
      - `/usr/lib/aarch64-linux-gnu/gentl/tis/libic4-gentl-u3v.cti` - Linux ARM64
      - `/usr/lib/x86_64-linux-gnu/gentl/tis/libic4-gentl-u3v.cti` - Linux x86_64

        · `C:/Program Files/The Imaging Source Europe GmbH/IC4 GenTL Driver for USB3Vision Devices/bin/ic4-gentl-u3v.cti` - Windows
    - XIMEA:
      * legacy cameras (FireWire, USB 2, older USB 3 camera families, like xiD and xiQ):
        · `/opt/XIMEA/lib/ximea.gentl.cti` - Linux
        · `C:/XIMEA/GenTL Producer/ximea.gentl.cti` - Windows
      * modern cameras (PCIe, newer USB 3 camera families, like xiC and xiJ):
        · `/opt/XIMEA/lib/ximea.gentl2.cti` - Linux
        · `C:/XIMEA/GenTL Producer/ximea.gentl2.cti` - Windows
- `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
- `max_buffers_queue_size` (default 2): maximum number of buffers to keep in acquisition queue
- `min_buffers_queue_size` (default `max_buffers_queue_size`-1): minimum number of buffers to keep 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): array of custom GenICam camera parameters, for example:
  - Basler:
    * `{ "BslColorSpace": "Off" }` - disables gamma correction, making acquired images linear, as expected for raw (e.g. `Bayer`) pixel formats
  - Baumer:
    * `{ "ColorTransformationAuto": "Off" }, { "BalanceWhiteAuto": "Off" }, { "GainSelector": "Red" }, { "Gain": 1.0 }, { "GainSelector": "GreenRed" }, { "Gain": 1.0 }, { "GainSelector": "GreenBlue" }, { "Gain": 1.0 }, { "GainSelector": "Blue" }, { "Gain": 1.0 }, { "GainSelector": "All" }` - disables in-camera white balance for it to be handled after acquisition, as it's usually done for `Bayer` pixel formats
  - The Imaging Source:
    * `{ "BlackLevel": 0.0 }` - sets black level to zero for correct image rendering
    * `{ "SoftwareTransformEnable": false }` - explicitly disables unneeded processing steps
    * `{ "BalanceWhiteAuto": "Off" }, { "BalanceRatioSelector": "Red" }, { "BalanceRatio": 1.0 }, { "BalanceRatioSelector": "Green" }, { "BalanceRatio": 1.0 }, { "BalanceRatioSelector": "Blue" }, { "BalanceRatio": 1.0 }` - disables in-camera white balance for it to be handled after acquisition, as it's usually done for `Bayer` pixel formats
- `black_level` (default 0): fallback black level, used only in case GenTL producer doesn't provide it
- `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): white balance settings, can be modified at runtime, can be specified in the following two ways (in order of precedence):

  - using color temperature:
    * `dcp_file`: path to DNG color profile file
    * `temperature` (default 5003): white balance temperature in kelvins
  - using white balance coefficients:
    * `r`, `g1`, `g2`, `b` (default 1.0): white balance gains for four Bayer channels (red, first green, second green, blue)
    * `g`: alternative way to specify both `g1` and `g2` at the same time, if they are equal (`g1` and `g2` take priority over this setting)
    * `r_off`, `g1_off`, `g2_off`, `b_off` (default 0): offsets for histogram stretch white balance algorithm (rarely used) for four Bayer channels (red, first green, second green, blue)
    * `g_off`: alternative way to specify both `g1_off` and `g2_off` at the same time, if they are equal (`g1_off` and `g2_off` take priority over this setting)

special parameters

Special parameters are read only. Their values can be read by calling `iff_get_params()`.

- `vendor_name`: camera manufacturer
- `device_type`: camera interface type
- `device_name`: camera model
- `serial_number`: camera serial number
- `min_exposure`: minimum camera exposure time in microseconds
- `max_exposure`: maximum camera exposure time in microseconds
- `min_gain`: minimum camera gain in dB
- `max_gain`: maximum camera gain in dB

4.1.2 `frame_importer`

Dispatches an externally filled buffer to the processing pipeline (see `iff_push_import_buffer()`). It should be used to pass frame data from user code across IFF SDK library boundaries.

JSON configuration:

```
{
    "id": "importer",
    "type": "frame_importer",
    "device_id": "cpu_dev",
    "max_processing_count": 2,
    "width": 3840,
    "height": 2160,
    "padding": 0,
    "format": "RGB8",
    "generate_timestamps": true
}
```

parameters

- `device_id`: device ID
- `padding` (default 0): row padding in bytes of the image
- `format`: image pixel format, see supported formats below
- `width`: image width in pixels
- `height`: image height in pixels

- `generate_timestamps` (default `true`): if set to `true` element will automatically generate `src_ts`, `ntp_ts` and `sequence_id` for each frame's metadata, see `iff_image_metadata`; otherwise, valid `src_ts`, `ntp_ts` and `sequence_id` must be provided via `metadata` parameter with each call to `iff_push_import_buffer()` function

supported formats

- `Mono8` - Monochrome 8-bit
- `Mono9` - Monochrome 9-bit unpacked
- `Mono10` - Monochrome 10-bit unpacked
- `Mono11` - Monochrome 11-bit unpacked
- `Mono12` - Monochrome 12-bit unpacked
- `Mono13` - Monochrome 13-bit unpacked
- `Mono14` - Monochrome 14-bit unpacked
- `Mono15` - Monochrome 15-bit unpacked
- `Mono16` - Monochrome 16-bit
- `Mono9p` - Monochrome 9-bit packed
- `Mono10p` - Monochrome 10-bit packed
- `Mono11p` - Monochrome 11-bit packed
- `Mono12p` - Monochrome 12-bit packed
- `Mono13p` - Monochrome 13-bit packed
- `Mono14p` - Monochrome 14-bit packed
- `Mono15p` - Monochrome 15-bit packed
- `RGB8` - Red-Green-Blue 8-bit
- `RGB9` - Red-Green-Blue 9-bit unpacked
- `RGB10` - Red-Green-Blue 10-bit unpacked
- `RGB11` - Red-Green-Blue 11-bit unpacked
- `RGB12` - Red-Green-Blue 12-bit unpacked
- `RGB13` - Red-Green-Blue 13-bit unpacked
- `RGB14` - Red-Green-Blue 14-bit unpacked
- `RGB15` - Red-Green-Blue 15-bit unpacked
- `RGB16` - Red-Green-Blue 16-bit
- `BGR8` - Blue-Green-Red 8-bit
- `BGR9` - Blue-Green-Red 9-bit unpacked
- `BGR10` - Blue-Green-Red 10-bit unpacked
- `BGR11` - Blue-Green-Red 11-bit unpacked
- `BGR12` - Blue-Green-Red 12-bit unpacked
- `BGR13` - Blue-Green-Red 13-bit unpacked
- `BGR14` - Blue-Green-Red 14-bit unpacked
- `BGR15` - Blue-Green-Red 15-bit unpacked
- `BGR16` - Blue-Green-Red 16-bit
- `RGBA8` - Red-Green-Blue-Alpha 8-bit
- `RGBA9` - Red-Green-Blue-Alpha 9-bit unpacked
- `RGBA10` - Red-Green-Blue-Alpha 10-bit unpacked
- `RGBA11` - Red-Green-Blue-Alpha 11-bit unpacked
- `RGBA12` - Red-Green-Blue-Alpha 12-bit unpacked

- `RGBA13` - Red-Green-Blue-Alpha 13-bit unpacked
- `RGBA14` - Red-Green-Blue-Alpha 14-bit unpacked
- `RGBA15` - Red-Green-Blue-Alpha 15-bit unpacked
- `RGBA16` - Red-Green-Blue-Alpha 16-bit
- `BGRA8` - Blue-Green-Red-Alpha 8-bit
- `BGRA9` - Blue-Green-Red-Alpha 9-bit unpacked
- `BGRA10` - Blue-Green-Red-Alpha 10-bit unpacked
- `BGRA11` - Blue-Green-Red-Alpha 11-bit unpacked
- `BGRA12` - Blue-Green-Red-Alpha 12-bit unpacked
- `BGRA13` - Blue-Green-Red-Alpha 13-bit unpacked
- `BGRA14` - Blue-Green-Red-Alpha 14-bit unpacked
- `BGRA15` - Blue-Green-Red-Alpha 15-bit unpacked
- `BGRA16` - Blue-Green-Red-Alpha 16-bit
- `BayerRG8` - Bayer Red-Green 8-bit
- `BayerRG9` - Bayer Red-Green 9-bit unpacked
- `BayerRG10` - Bayer Red-Green 10-bit unpacked
- `BayerRG11` - Bayer Red-Green 11-bit unpacked
- `BayerRG12` - Bayer Red-Green 12-bit unpacked
- `BayerRG13` - Bayer Red-Green 13-bit unpacked
- `BayerRG14` - Bayer Red-Green 14-bit unpacked
- `BayerRG15` - Bayer Red-Green 15-bit unpacked
- `BayerRG16` - Bayer Red-Green 16-bit
- `BayerBG8` - Bayer Blue-Green 8-bit
- `BayerBG9` - Bayer Blue-Green 9-bit unpacked
- `BayerBG10` - Bayer Blue-Green 10-bit unpacked
- `BayerBG11` - Bayer Blue-Green 11-bit unpacked
- `BayerBG12` - Bayer Blue-Green 12-bit unpacked
- `BayerBG13` - Bayer Blue-Green 13-bit unpacked
- `BayerBG14` - Bayer Blue-Green 14-bit unpacked
- `BayerBG15` - Bayer Blue-Green 15-bit unpacked
- `BayerBG16` - Bayer Blue-Green 16-bit
- `BayerGR8` - Bayer Green-Red 8-bit
- `BayerGR9` - Bayer Green-Red 9-bit unpacked
- `BayerGR10` - Bayer Green-Red 10-bit unpacked
- `BayerGR11` - Bayer Green-Red 11-bit unpacked
- `BayerGR12` - Bayer Green-Red 12-bit unpacked
- `BayerGR13` - Bayer Green-Red 13-bit unpacked
- `BayerGR14` - Bayer Green-Red 14-bit unpacked
- `BayerGR15` - Bayer Green-Red 15-bit unpacked
- `BayerGR16` - Bayer Green-Red 16-bit
- `BayerGB8` - Bayer Green-Blue 8-bit
- `BayerGB9` - Bayer Green-Blue 9-bit unpacked
- `BayerGB10` - Bayer Green-Blue 10-bit unpacked

- `BayerGB11` - Bayer Green-Blue 11-bit unpacked
- `BayerGB12` - Bayer Green-Blue 12-bit unpacked
- `BayerGB13` - Bayer Green-Blue 13-bit unpacked
- `BayerGB14` - Bayer Green-Blue 14-bit unpacked
- `BayerGB15` - Bayer Green-Blue 15-bit unpacked
- `BayerGB16` - Bayer Green-Blue 16-bit
- `BayerRG9p` - Bayer Red-Green 9-bit packed
- `BayerRG10p` - Bayer Red-Green 10-bit packed
- `BayerRG11p` - Bayer Red-Green 11-bit packed
- `BayerRG12p` - Bayer Red-Green 12-bit packed
- `BayerRG13p` - Bayer Red-Green 13-bit packed
- `BayerRG14p` - Bayer Red-Green 14-bit packed
- `BayerRG15p` - Bayer Red-Green 15-bit packed
- `BayerBG9p` - Bayer Blue-Green 9-bit packed
- `BayerBG10p` - Bayer Blue-Green 10-bit packed
- `BayerBG11p` - Bayer Blue-Green 11-bit packed
- `BayerBG12p` - Bayer Blue-Green 12-bit packed
- `BayerBG13p` - Bayer Blue-Green 13-bit packed
- `BayerBG14p` - Bayer Blue-Green 14-bit packed
- `BayerBG15p` - Bayer Blue-Green 15-bit packed
- `BayerGR9p` - Bayer Green-Red 9-bit packed
- `BayerGR10p` - Bayer Green-Red 10-bit packed
- `BayerGR11p` - Bayer Green-Red 11-bit packed
- `BayerGR12p` - Bayer Green-Red 12-bit packed
- `BayerGR13p` - Bayer Green-Red 13-bit packed
- `BayerGR14p` - Bayer Green-Red 14-bit packed
- `BayerGR15p` - Bayer Green-Red 15-bit packed
- `BayerGB9p` - Bayer Green-Blue 9-bit packed
- `BayerGB10p` - Bayer Green-Blue 10-bit packed
- `BayerGB11p` - Bayer Green-Blue 11-bit packed
- `BayerGB12p` - Bayer Green-Blue 12-bit packed
- `BayerGB13p` - Bayer Green-Blue 13-bit packed
- `BayerGB14p` - Bayer Green-Blue 14-bit packed
- `BayerGB15p` - Bayer Green-Blue 15-bit packed
- `YV12` - 8-bit planar YVU 4:2:0 subsampling
- `I420_10LE` - 10-bit planar YUV 4:2:0 subsampling
- `NV12` - 8-bit semi-planar YUV 4:2:0 subsampling
- `P010` - 10-bit semi-planar YUV 4:2:0 subsampling
- `HistogramMono256Int`
- `HistogramMono512Int`
- `HistogramMono1024Int`
- `HistogramMono2048Int`
- `HistogramMono4096Int`

- HistogramMono8192Int
- HistogramMono16384Int
- HistogramMono32768Int
- HistogramMono65536Int
- HistogramRGB256Int
- Histogram3Bayer256Int
- Histogram3Bayer512Int
- Histogram3Bayer1024Int
- Histogram3Bayer2048Int
- Histogram3Bayer4096Int
- Histogram3Bayer8192Int
- Histogram3Bayer16384Int
- Histogram3Bayer32768Int
- Histogram3Bayer65536Int
- Histogram4Bayer256Int
- Histogram4Bayer512Int
- Histogram4Bayer1024Int
- Histogram4Bayer2048Int
- Histogram4Bayer4096Int
- Histogram4Bayer8192Int
- Histogram4Bayer16384Int
- Histogram4Bayer32768Int
- Histogram4Bayer65536Int
- HistogramParade256Int

For format description see GenICam Pixel Format Naming Convention (PFNC) Version 2.4, YUV formats section of `export_to_device` documentation and Histogram formats section of `histogram` documentation.

4.1.3 `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,
    "padding": 0,
    "format": "BayerRG8",
    "fps": 30.0,
```

```
    "loop_images": false,
    "io_timer_interval": 10,
    "max_cached_images_count": 2,
    "wb": {
        "dcp_file": "color_profile.dcp",
        "temperature": 5003,
        "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",
    "template_params": {
        "aperture": 1.4
    },
    "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
- `padding` (default 0): row padding in bytes of images stored in files
- `format`: pixel format of images stored in files, see supported formats below
- `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): white balance settings, can be modified at runtime, can be specified in the following two ways (in order of precedence):
  - using color temperature:
    - `dcp_file`: path to DNG color profile file
    - `temperature` (default 5003): white balance temperature in kelvins
  - using white balance coefficients:
    - `r`, `g1`, `g2`, `b` (default 1.0): white balance gains for four Bayer channels (red, first green, second green, blue)
    - `g`: alternative way to specify both `g1` and `g2` at the same time, if they are equal (`g1` and `g2` take priority over this setting)
    - `r_off`, `g1_off`, `g2_off`, `b_off` (default 0): offsets for histogram stretch white balance algorithm (rarely used) for four Bayer channels (red, first green, second green, blue)
    - `g_off`: alternative way to specify both `g1_off` and `g2_off` at the same time, if they are equal (`g1_off` and `g2_off` take priority over this setting)

- `filename_template` (optional): string in {fmt} library format to use as filename template, refer to description of this parameter for frames_writer
- `template_params` (optional): additional static parameters (string or number) for `filename_template`
- `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;
- `src_ts` timestamp after the first loop over all files (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.

---

supported formats

- `Mono8` - Monochrome 8-bit
- `Mono9` - Monochrome 9-bit unpacked
- `Mono10` - Monochrome 10-bit unpacked
- `Mono11` - Monochrome 11-bit unpacked
- `Mono12` - Monochrome 12-bit unpacked
- `Mono13` - Monochrome 13-bit unpacked
- `Mono14` - Monochrome 14-bit unpacked
- `Mono15` - Monochrome 15-bit unpacked
- `Mono16` - Monochrome 16-bit
- `Mono9p` - Monochrome 9-bit packed
- `Mono10p` - Monochrome 10-bit packed
- `Mono11p` - Monochrome 11-bit packed
- `Mono12p` - Monochrome 12-bit packed
- `Mono13p` - Monochrome 13-bit packed
- `Mono14p` - Monochrome 14-bit packed
- `Mono15p` - Monochrome 15-bit packed
- `RGB8` - Red-Green-Blue 8-bit
- `RGB9` - Red-Green-Blue 9-bit unpacked
- `RGB10` - Red-Green-Blue 10-bit unpacked
- `RGB11` - Red-Green-Blue 11-bit unpacked
- `RGB12` - Red-Green-Blue 12-bit unpacked

- `RGB13` - Red-Green-Blue 13-bit unpacked
- `RGB14` - Red-Green-Blue 14-bit unpacked
- `RGB15` - Red-Green-Blue 15-bit unpacked
- `RGB16` - Red-Green-Blue 16-bit
- `BGR8` - Blue-Green-Red 8-bit
- `BGR9` - Blue-Green-Red 9-bit unpacked
- `BGR10` - Blue-Green-Red 10-bit unpacked
- `BGR11` - Blue-Green-Red 11-bit unpacked
- `BGR12` - Blue-Green-Red 12-bit unpacked
- `BGR13` - Blue-Green-Red 13-bit unpacked
- `BGR14` - Blue-Green-Red 14-bit unpacked
- `BGR15` - Blue-Green-Red 15-bit unpacked
- `BGR16` - Blue-Green-Red 16-bit
- `RGBA8` - Red-Green-Blue-Alpha 8-bit
- `RGBA9` - Red-Green-Blue-Alpha 9-bit unpacked
- `RGBA10` - Red-Green-Blue-Alpha 10-bit unpacked
- `RGBA11` - Red-Green-Blue-Alpha 11-bit unpacked
- `RGBA12` - Red-Green-Blue-Alpha 12-bit unpacked
- `RGBA13` - Red-Green-Blue-Alpha 13-bit unpacked
- `RGBA14` - Red-Green-Blue-Alpha 14-bit unpacked
- `RGBA15` - Red-Green-Blue-Alpha 15-bit unpacked
- `RGBA16` - Red-Green-Blue-Alpha 16-bit
- `BGRA8` - Blue-Green-Red-Alpha 8-bit
- `BGRA9` - Blue-Green-Red-Alpha 9-bit unpacked
- `BGRA10` - Blue-Green-Red-Alpha 10-bit unpacked
- `BGRA11` - Blue-Green-Red-Alpha 11-bit unpacked
- `BGRA12` - Blue-Green-Red-Alpha 12-bit unpacked
- `BGRA13` - Blue-Green-Red-Alpha 13-bit unpacked
- `BGRA14` - Blue-Green-Red-Alpha 14-bit unpacked
- `BGRA15` - Blue-Green-Red-Alpha 15-bit unpacked
- `BGRA16` - Blue-Green-Red-Alpha 16-bit
- `BayerRG8` - Bayer Red-Green 8-bit
- `BayerRG9` - Bayer Red-Green 9-bit unpacked
- `BayerRG10` - Bayer Red-Green 10-bit unpacked
- `BayerRG11` - Bayer Red-Green 11-bit unpacked
- `BayerRG12` - Bayer Red-Green 12-bit unpacked
- `BayerRG13` - Bayer Red-Green 13-bit unpacked
- `BayerRG14` - Bayer Red-Green 14-bit unpacked
- `BayerRG15` - Bayer Red-Green 15-bit unpacked
- `BayerRG16` - Bayer Red-Green 16-bit
- `BayerBG8` - Bayer Blue-Green 8-bit
- `BayerBG9` - Bayer Blue-Green 9-bit unpacked
- `BayerBG10` - Bayer Blue-Green 10-bit unpacked

- `BayerBG11` - Bayer Blue-Green 11-bit unpacked
- `BayerBG12` - Bayer Blue-Green 12-bit unpacked
- `BayerBG13` - Bayer Blue-Green 13-bit unpacked
- `BayerBG14` - Bayer Blue-Green 14-bit unpacked
- `BayerBG15` - Bayer Blue-Green 15-bit unpacked
- `BayerBG16` - Bayer Blue-Green 16-bit
- `BayerGR8` - Bayer Green-Red 8-bit
- `BayerGR9` - Bayer Green-Red 9-bit unpacked
- `BayerGR10` - Bayer Green-Red 10-bit unpacked
- `BayerGR11` - Bayer Green-Red 11-bit unpacked
- `BayerGR12` - Bayer Green-Red 12-bit unpacked
- `BayerGR13` - Bayer Green-Red 13-bit unpacked
- `BayerGR14` - Bayer Green-Red 14-bit unpacked
- `BayerGR15` - Bayer Green-Red 15-bit unpacked
- `BayerGR16` - Bayer Green-Red 16-bit
- `BayerGB8` - Bayer Green-Blue 8-bit
- `BayerGB9` - Bayer Green-Blue 9-bit unpacked
- `BayerGB10` - Bayer Green-Blue 10-bit unpacked
- `BayerGB11` - Bayer Green-Blue 11-bit unpacked
- `BayerGB12` - Bayer Green-Blue 12-bit unpacked
- `BayerGB13` - Bayer Green-Blue 13-bit unpacked
- `BayerGB14` - Bayer Green-Blue 14-bit unpacked
- `BayerGB15` - Bayer Green-Blue 15-bit unpacked
- `BayerGB16` - Bayer Green-Blue 16-bit
- `BayerRG9p` - Bayer Red-Green 9-bit packed
- `BayerRG10p` - Bayer Red-Green 10-bit packed
- `BayerRG11p` - Bayer Red-Green 11-bit packed
- `BayerRG12p` - Bayer Red-Green 12-bit packed
- `BayerRG13p` - Bayer Red-Green 13-bit packed
- `BayerRG14p` - Bayer Red-Green 14-bit packed
- `BayerRG15p` - Bayer Red-Green 15-bit packed
- `BayerBG9p` - Bayer Blue-Green 9-bit packed
- `BayerBG10p` - Bayer Blue-Green 10-bit packed
- `BayerBG11p` - Bayer Blue-Green 11-bit packed
- `BayerBG12p` - Bayer Blue-Green 12-bit packed
- `BayerBG13p` - Bayer Blue-Green 13-bit packed
- `BayerBG14p` - Bayer Blue-Green 14-bit packed
- `BayerBG15p` - Bayer Blue-Green 15-bit packed
- `BayerGR9p` - Bayer Green-Red 9-bit packed
- `BayerGR10p` - Bayer Green-Red 10-bit packed
- `BayerGR11p` - Bayer Green-Red 11-bit packed
- `BayerGR12p` - Bayer Green-Red 12-bit packed
- `BayerGR13p` - Bayer Green-Red 13-bit packed

- `BayerGR14p` - Bayer Green-Red 14-bit packed
- `BayerGR15p` - Bayer Green-Red 15-bit packed
- `BayerGB9p` - Bayer Green-Blue 9-bit packed
- `BayerGB10p` - Bayer Green-Blue 10-bit packed
- `BayerGB11p` - Bayer Green-Blue 11-bit packed
- `BayerGB12p` - Bayer Green-Blue 12-bit packed
- `BayerGB13p` - Bayer Green-Blue 13-bit packed
- `BayerGB14p` - Bayer Green-Blue 14-bit packed
- `BayerGB15p` - Bayer Green-Blue 15-bit packed
- `YV12` - 8-bit planar YVU 4:2:0 subsampling
- `I420_10LE` - 10-bit planar YUV 4:2:0 subsampling
- `NV12` - 8-bit semi-planar YUV 4:2:0 subsampling
- `P010` - 10-bit semi-planar YUV 4:2:0 subsampling

For format description see GenICam Pixel Format Naming Convention (PFNC) Version 2.4 and YUV formats section of `export_to_device` `cuda_processor` filter documentation.

4.1.4 `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` (default)
- `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.

4.1.5 `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" : [
        { "override_enable": 1 },
        { "bypass_mode": 0 }
    ],
    "black_level": 0,
    "exposure": 10000,
    "fps": 60.0,
    "gain": 0.0,
    "linear_gain": false,
    "wb": {
        "dcp_file": "color_profile.dcp",
        "temperature": 5003,
        "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, recommended value:
  - MIPI and GMSL cameras on NVIDIA Jetson - sum of `max_processing_count` settings of this `v4l2cam` element and all elements directly connected to its output (default zero value causes increased CPU load and latency)

  - other cameras - default zero value should work fine in most cases, but formula above can still be used when in doubt
- `min_buffers_queue_size` (default `max_processing_count`-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): array of custom camera control parameters, names can be looked up in `v4l2-ctl -l` output, for example:
  - MIPI and GMSL cameras on NVIDIA Jetson:
    * `{ "override_enable": 1 }` - ensures correct synchronization of settings (like exposure and gain) between camera and OS
    * `{ "bypass_mode": 0 }` - restores image acquisition through Video4Linux2 interface in case `libargus` applications (utilizing NVIDIA hardware ISP) were used in the same boot session
- `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 dB, not modified by default
- `linear_gain` (default false): whether Video4Linux camera gain control is in linear scale and has to be converted to logarithmic dB values
- `wb` (optional): white balance settings, can be modified at runtime, can be specified in the following two ways (in order of precedence):
  - using color temperature:
    * `dcp_file`: path to DNG color profile file
    * `temperature` (default 5003): white balance temperature in kelvins
  - using white balance coefficients:
    * `r`, `g1`, `g2`, `b` (default 1.0): white balance gains for four Bayer channels (red, first green, second green, blue)
    * `g`: alternative way to specify both `g1` and `g2` at the same time, if they are equal (`g1` and `g2` take priority over this setting)
    * `r_off`, `g1_off`, `g2_off`, `b_off` (default 0): offsets for histogram stretch white balance algorithm (rarely used) for four Bayer channels (red, first green, second green, blue)
    * `g_off`: alternative way to specify both `g1_off` and `g2_off` at the same time, if they are equal (`g1_off` and `g2_off` take priority over this setting)

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.

---

special parameters
Special parameters are read only. Their values can be read by calling `iff_get_params()`.

- `vendor_name`: camera manufacturer
- `device_name`: camera model
- `min_exposure`: minimum camera exposure time in microseconds
- `max_exposure`: maximum camera exposure time in microseconds
- `min_gain`: minimum camera gain in dB
- `max_gain`: maximum camera gain in dB

4.1.6 `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_white_balance": false,
    "wb": {
        "dcp_file": "color_profile.dcp",
```

```
        "temperature": 5003,
        "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` (default)
  - `ERROR`
  - `FATAL`
  - `DISABLED`
- `auto_bandwidth_calculation` (default true): whether to enable auto bandwidth calculation in xiAPI
- `image_format` (default "RAW8"): camera output xiAPI image data format, one of the following:
  - `MONO8`
  - `MONO16`
  - `RAW8` (default)
  - `RAW16`
  - `RGB24`
  - `RGB32`
  - `RGB48`
  - `RGB64`
  - `TRANSPORT_DATA`
- `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): array of 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_white_balance` (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): white balance settings, can be modified at runtime, can be specified in the following two ways (in order of precedence):
  - using color temperature:
    - `dcp_file`: path to DNG color profile file
    - `temperature` (default 5003): white balance temperature in kelvins
  - using white balance coefficients:
    - `r`, `g1`, `g2`, `b` (default 1.0): white balance gains for four Bayer channels (red, first green, second green, blue)
    - `g`: alternative way to specify both `g1` and `g2` at the same time, if they are equal (`g1` and `g2` take priority over this setting)
    - `r_off`, `g1_off`, `g2_off`, `b_off` (default 0): offsets for histogram stretch white balance algorithm (rarely used) for four Bayer channels (red, first green, second green, blue)
    - `g_off`: alternative way to specify both `g1_off` and `g2_off` at the same time, if they are equal (`g1_off` and `g2_off` take priority over this setting)

special parameters

Special parameters are read only. Their values can be read by calling `iff_get_params()`.

- `bad_pixel_map`: bad pixel map
- `temperature`: camera temperature in degrees Celsius
- `vendor_name`: camera manufacturer
- `device_type`: camera interface type
- `device_name`: camera model
- `serial_number`: camera serial number
- `min_exposure`: minimum camera exposure time in microseconds
- `max_exposure`: maximum camera exposure time in microseconds
- `min_gain`: minimum camera gain in dB
- `max_gain`: maximum camera gain in dB

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 commands:

- `on` - makes sink start processing images
  execution result format:

```
{ "success": true }
```

- `off` - makes sink stop processing images
  execution result format:

```
{ "success": true }
```

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

Any sink has those two callbacks:

- `on_started` - called when the sink is turned on
- `on_stopped` - called when the sink is turned off

Both of these callbacks return empty JSON. See `iff_set_callback()` for information on how to set callback for an element.

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": false,
    "awb_enabled": false,
    "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,
    "ctrl_latency": 3,
    "add_frames": 0,
    "min_exposure": 100,
    "max_exposure": 0,
    "min_gain": 0.0,
    "max_gain": 0.0,
    "gain_step": 1.0,
    "gain_priority": 0.0,
    "exposure_margin": 0.05,
```

```
    "hdr_threshold_low": 1.0,
    "hdr_threshold_high": 1.0,
    "ev_correction": 0.0,
    "hdr_median_ev": -3.0
}
```

formula

$whitepoint = bins - 1$

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

$j \in \mathrm{I}$

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

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

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

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

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

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

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

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

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

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

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

$$sum\_corr_i = sum_i - corr_i$$

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

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

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

$$out\_off_i = 0$$

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

histogram stretch white balance

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

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

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

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

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

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

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

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

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

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

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

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 settings.

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

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

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

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

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

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

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

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

$$min\_gain\_corr = 10^{\frac{min\_gain - gain}{20}} \qquad \text{(j)}$$

where $gain$ is gain value taken from image metadata

$$min\_time\_corr = \frac{min\_exposure}{exposure} \qquad \text{(k)}$$

where $exposure$ is exposure time taken from image metadata

$$time\_corr = \left(\frac{exp\_corr}{min\_gain\_corr \cdot min\_time\_corr} \right)^{1 - gain\_priority} \qquad \text{(l)}$$

$$gain\_corr = \begin{cases} \frac{exp\_corr}{min\_gain\_corr \cdot min\_time\_corr} & time\_corr < 1 < \frac{max\_exposure}{min\_exposure} \\ \frac{exp\_corr}{min\_gain\_corr \cdot min\_time\_corr \cdot time\_corr} & 1 < time\_corr < \frac{max\_exposure}{min\_exposure} \\ \frac{exp\_corr}{min\_gain\_corr \cdot min\_time\_corr \cdot \frac{max\_exposure}{min\_exposure}} & 1 < \frac{max\_exposure}{min\_exposure} < time\_corr \\ \frac{exp\_corr}{min\_gain\_corr} & \text{otherwise} \end{cases} \qquad \text{(m)}$$

$$target\_gain = min\_gain + \left\lceil \frac{20 \cdot \log_{10} gain\_corr}{gain\_step} - \frac{1}{2} \right\rceil \cdot gain\_step \qquad \text{(n)}$$

$$out\_gain = \begin{cases} min\_gain & target\_gain \leq min\_gain < max\_gain \\ target\_gain & min\_gain < target\_gain < max\_gain \\ max\_gain & min\_gain < max\_gain \leq target\_gain \\ gain & min\_gain \geq max\_gain \end{cases} \qquad \text{(o)}$$

$$target\_exp = exposure \cdot exp\_corr \cdot 10^{\frac{gain - out\_gain}{20}} \qquad \text{(p)}$$

$$set\_exp = \begin{cases} min\_exposure & target\_exp \leq min\_exposure < max\_exposure \\ target\_exp & min\_exposure < target\_exp < max\_exposure \\ max\_exposure & min\_exposure < max\_exposure \leq target\_exp \\ exposure & min\_exposure \geq max\_exposure \end{cases} \qquad \text{(q)}$$

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

(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 exposure correction factor. (j) calculates the lowest exposure correction, that can be done by changing only gain value. (k) calculates the lowest exposure correction, that can be done by changing only exposure time. (l) estimates the portion of exposure correction relative to the lowest achievable exposure to be done by exposure time control. (m) calculates the portion of exposure correction relative to the lowest achievable exposure to be done by gain value control. (n) calculates target gain value. (o) clamps calculated gain value to the defined boundaries. (p) calculates target exposure time. (q) clamps calculated exposure time to the defined boundaries. (r) checks if target exposure time falls within specified margins from the current value 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
- `ctrl_latency` (default 3): how many frames to skip after adjusting the exposure before evaluating it again
- `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 microseconds that is going to be set
- `max_exposure` (default 0): maximum exposure time in microseconds that is going to be set
- `min_gain` (default 0.0): minimum gain in dB that is going to be set
- `max_gain` (default 0.0): maximum gain in dB that is going to be set
- `gain_step` (default 1.0): granularity of camera gain control in dB
- `gain_priority` (default 0.0): prefer changing exposure time (lower values) or gain (higher values), valid range from 0.0 to 1.0
- `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, can be modified at runtime
- `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` - called when white balance 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
    }
}
```

- `wb`: calculated white balance parameters

- `exposure_callback` - called when exposure and gain parameters have been calculated by the element

  exposure_callback data format:

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

  - `exposure`: calculated exposure time in microseconds
  - `gain`: calculated gain in dB

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": false,
    "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 false): 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_callback()`). 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",
    "async_mode": false
}
```

parameters

- `device_id`: device ID
- `async_mode` (default `false`): if set to `false` buffer will be automatically released and no longer valid after export callback returns (see `iff_set_export_callback()`); otherwise, buffer must be released manually when it is no longer needed by calling `iff_release_buffer()` function

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",
    "template_params": {
        "aperture": 1.4
    },
    "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
  - `src_ts` - frame timestamp (usually in micro-seconds) provided by camera or other source
  - `ntp_ts` - frame NTP UTC date and time, use `strftime`-like formatting
  - `ntp_ts_local` - frame NTP local date and time, use `strftime`-like formatting
  - `ntp_ts_us` - sub-second part of frame NTP timestamp in micro-seconds
  - `utc_time` - frame NTP UTC date and time in ISO 8601 format (same as `{ntp_ts:%Y%m%dT%H%M%S}.{ntp_ts_us:06}Z`)
  - `black_level` - frame black level
  - `exposure` - frame exposure time
  - `gain` - frame gain
  - `sequence_id` - frame sequence id
- `template_params` (optional): additional static parameters (string or number) for `filename_template`
- `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

callbacks

- `frame_written_callback` - called for every frame
  frame_written_callback data format:

```
{
    "success": true
}
```

  - `success`: whether the frame was successfully written
- `write_complete_callback` - called when the element is turned off
  write_complete_callback data format:

```
{
    "written_frames_count": 42
}
```

  - `written_frames_count`: number of written (successfully or not) frames since the last time the element was turned on

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` and `Monopmsb` - LinearRaw DNG
- `Bayer` and `Bayerpmsb` - 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": "",
    "base_iso": 0.0,
    "baseline_exposure": 0.0,
    "frame_rate": 0.0,
    "base_frame_rate": "30,25,24",
    "t_stop": 0.0,
    "reel_name": "",
    "camera_label": "",
    "orientation": "normal",
    "wb_preapplied": false,
    "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
- `base_iso` (default 0.0): base ISO rating of the camera (with gain set to zero), that will be used to compute ISOSpeedRatings TIFF tag value, if not zero
- `baseline_exposure` (default 0.0): rational number, that will be written to BaselineExposure DNG tag, if not zero
- `frame_rate` (default 0.0): rational number, that will be written to FrameRate CinemaDNG tag, if not zero
- `base_frame_rate` (default ”30,25,24”): one of the following strings, which specifies the order in which base (super) frame rates are checked to be a factor of `frame_rate` when creating a SMPTE time code for TimeCodes CinemaDNG tag:
  - ”24,25,30”
  - ”24,30,25”
  - ”25,24,30”
  - ”25,30,24”
  - ”30,24,25”
  - ”30,25,24” (default)
- `t_stop` (default 0.0): rational number, that will be written to TStop CinemaDNG tag, if not zero
- `reel_name` (default ””): string, that will be written to ReelName CinemaDNG tag, if not empty
- `camera_label` (default ””): string, that will be written to CameraLabel CinemaDNG tag, if not empty
- `orientation` (default ”normal”): value, that will be written to Orientation TIFF tag, specified as an integer number or as one of the following strings:
  - ”top_left” (1) - default
  - ”normal” (1) - default
  - ”top_right” (2)
  - ”mirrored_horiz” (2)
  - ”bottom_right” (3)
  - ”rotated_180” (3)
  - ”bottom_left” (4)
  - ”mirrored_vert” (4)
  - ”left_top” (5)
  - ”right_top” (6)
  - ”rotated_cw_90” (6)
  - ”right_bottom” (7)
  - ”left_bottom” (8)
  - ”rotated_ccw_90” (8)
  - ”unknown” (9)
- `wb_preapplied` (default false): whether white balance has been already applied to the incoming `Bayer` image (ColorMatrix and AsShotNeutral DNG tags are adjusted accordingly in this case)
- `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)
    * ”WarmWhiteFluorescent” (16)
    * ”StandardLightA” (17)
    * ”StandardLightB” (18)
    * ”StandardLightC” (19)
    * ”D55” (20)
    * ”D65” (21)
    * ”D75” (22)
    * ”D50” (23) - default
    * ”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.

commands
All frames_writer commands are supported.

callbacks
All frames_writer callbacks are supported.

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)
- SMPTE ST 331:2011 ”Element and Metadata Definitions for the SDTI-CP”
- SMPTE ST 12-1:2014 ”Time and Control Code”
- SMPTE ST 309:2012 ”Transmission of Date and Time Zone Information in Binary Groups of Time and Control Code”

4.2.6 `exr_writer`

Writes each received linear RGB image to a separate EXR file in the given directory.

JSON configuration:

```
{
    "id": "writer",
    "type": "exr_writer",
    "max_processing_count": 2,
    "autostart": false,
    "cpu_device_id": "cpu_dev",
    "base_directory": "saved_frames",
    "filename_template": "{sequence_number:06}.exr",
    "template_params": {
        "aperture": 1.4
    },
    "data_format": "half",
    "compression": "PIZ",
    "zip_compression_level": 4,
    "dwa_compression": 45.0,
    "num_threads": 0,
    "colorspace": "Rec709",
    "temperature": 0.0,
    "make": "",
    "model": "",
    "serial_number": "",
    "copyright": "",
    "description": "",
    "base_iso": 0.0,
    "baseline_exposure": 0.0,
    "frame_rate": 0.0,
    "base_frame_rate": "30,25,24",
    "t_stop": 0.0,
```

```
    "reel_name": "",
    "camera_label": ""
}
```

parameters

- `cpu_device_id`: CPU device ID
- `base_directory` (default "saved_frames"): path to the directory to save files in
- `filename_template` (default "{sequence_number:06}.exr"): 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
  - `src_ts` - frame timestamp (usually in micro-seconds) provided by camera or other source
  - `ntp_ts` - frame NTP UTC date and time, use `strftime`-like formatting
  - `ntp_ts_local` - frame NTP local date and time, use `strftime`-like formatting
  - `ntp_ts_us` - sub-second part of frame NTP timestamp in micro-seconds
  - `utc_time` - frame NTP UTC date and time in ISO 8601 format (same as `{ntp_ts:%Y%m%dT%H%M%S}.{ntp_ts_us:06}Z`)
  - `black_level` - frame black level
  - `exposure` - frame exposure time
  - `gain` - frame gain
  - `sequence_id` - frame sequence id
- `template_params` (optional): additional static parameters (string or number) for `filename_template`
- `data_format` (default "half"): data storage format of written pixels, one of the following:
  - `half` (default) - 16-bit floating-point numbers
  - `float` - 32-bit floating-point numbers
- `compression` (default "PIZ"): compression algorithm, one of the following:
  - `NO` - no compression
  - `RLE` - run length encoding
  - `ZIPS` - zlib compression, one scan-line at a time
  - `ZIP` - zlib compression, in blocks of 16 scan-lines
  - `PIZ` (default) - PIZ-based wavelet compression
  - `PXR24` - lossy 24-bit float compression
  - `B44` - lossy 4-by-4 pixel block compression, fixed compression rate
  - `B44A` - lossy 4-by-4 pixel block compression, flat fields are compressed more
  - `DWAA` - lossy DCT-based compression, in blocks of 32 scan-lines, more efficient for partial buffer access
  - `DWAB` - lossy DCT-based compression, in blocks of 256 scan-lines, more efficient space-wise and faster to decode full frames than `DWAA`
- `zip_compression_level` (default 4): compression level setting used in `ZIPS`, `ZIP`, `DWAA` and `DWAB` algorithms, ranging from 0 to 9 (higher values result in smaller files)
- `dwa_compression_level` (default 45.0): compression level setting used in `DWAA` and `DWAB` algorithms, ranging from 0.0 to 100.0 (higher values result in smaller files)

- `num_threads` (default 0): number of worker threads, non-positive value means auto-detect (using `std::thread::hardware_concurrency()`)
- `colorspace` (default "Rec709"): color space name, used to fill `chromaticities` and `adoptedNeutral` attributes, one of the following:
  - `ACES`
  - `ACEScg`
  - `DisplayP3`
  - `ProPhotoRGB`
  - `Rec709` (default) - same as sRGB
  - `Rec2020`
- `temperature` (default 0.0): number, that will be written to `cameraCCTSetting` attribute, if positive
- `make` (default ""): string, that will be written to `cameraMake` and `cameraUuid` attributes, if not empty
- `model` (default ""): string, that will be written to `cameraModel` and `cameraUuid` attributes, if not empty
- `serial_number` (default ""): string, that will be written to `cameraSerialNumber` and `cameraUuid` attributes, if not empty
- `copyright` (default ""): string, that will be written to `owner` attribute, if not empty
- `description` (default ""): string, that will be written to `comments` attribute, if not empty
- `base_iso` (default 0.0): base ISO rating of the camera (with gain set to zero), that will be used to compute `isoSpeed` attribute value, if positive
- `baseline_exposure` (default 0.0): exposure compensation setting in EV units, that will be used for scaling of output values (by default the output range is from 0.0 to 1.0)
- `frame_rate` (default 0.0): number, that will be written to `captureRate` and `framesPerSecond` attributes and will be used to calculate `shutterAngle` attribute value, if positive
- `base_frame_rate` (default "30,25,24"): one of the following strings, which specifies the order in which base (super) frame rates are checked to be a factor of `frame_rate` when creating a SMPTE time code for `timeCode` attribute:
  - "24,25,30"
  - "24,30,25"
  - "25,24,30"
  - "25,30,24"
  - "30,24,25"
  - "30,25,24" (default)
- `t_stop` (default 0.0): number, that will be written to `tStop` attribute, if positive
- `reel_name` (default ""): string, that will be written to `reelName` attribute, if not empty
- `camera_label` (default ""): string, that will be written to `cameraLabel` attribute, if not empty

Other metadata tags, like exposure time (`expTime`) and capture date (`capDate`), are filled from image metadata.

commands

All frames_writer commands are supported.

callbacks

All frames_writer callbacks are supported.

references

- OpenEXR Standard Attributes
- SMPTE ST 331:2011 "Element and Metadata Definitions for the SDTI-CP"
- SMPTE ST 12-1:2014 "Time and Control Code"
- SMPTE ST 309:2012 "Transmission of Date and Time Zone Information in Binary Groups of Time and Control Code"

4.2.7 `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
  metadata parameter data format:

```
{
    "metadata": [
        {
            "frame": 0,
            "sequence_id": 2,
            "ntp_ts": 16832616755504369933,
            "rtp_ts": 1374027318,
            "unix_ts": 1710160193.562993,
            "src_ts": 11,
            "black_level": 0,
            "exposure": 0,
            "gain": 0.0,
            "offset_x": 0,
            "offset_y": 0,
            "wb_b": 1.0,
            "wb_b_off": 0.0,
            "wb_g1": 1.0,
            "wb_g1_off": 0.0,
            "wb_g2": 1.0,
            "wb_g2_off": 0.0,
            "wb_r": 1.0,
            "wb_r_off": 0.0
        },
        {
            "frame": 1,
            "sequence_id": 2,
            "ntp_ts": 16832616755934335837,
            "rtp_ts": 1374036328,
            "unix_ts": 1710160193.6631024,
            "src_ts": 12,
            "black_level": 0,
            "exposure": 0,
```

```
            "gain": 0.0,
            "offset_x": 0,
            "offset_y": 0,
            "wb_b": 1.0,
            "wb_b_off": 0.0,
            "wb_g1": 1.0,
            "wb_g1_off": 0.0,
            "wb_g2": 1.0,
            "wb_g2_off": 0.0,
            "wb_r": 1.0,
            "wb_r_off": 0.0
        }
    ]
}
```

- `frame`: image sequence number
- `sequence_id`: ID of a dispatch session within which given image was dispatched, provided by source
- `ntp_ts`: image timestamp in NTP format (see RFC 5905)
- `rtp_ts`: image timestamp as it is transmitted in RTP header
- `unix_ts`: image timestamp as time in seconds since UNIX epoch
- `src_ts`: image timestamp provided by source
- `black_level`: image black level
- `exposure`: image exposure time in microseconds
- `gain`: image gain in dB
- `offset_x`: horizontal offset of ROI or crop position
- `offset_y`: vertical offset of ROI or crop position
- image white balance coefficients:
  * `wb_b_off`
  * `wb_g1`
  * `wb_g1_off`
  * `wb_g2`
  * `wb_g2_off`
  * `wb_r`
  * `wb_r_off`

4.2.8 `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 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 `NV12` (desktop GPU), `YV12` (Jetson), `P010` (desktop GPU and Jetson), `Mono8` (Jetson) and `Mono16` (Jetson) images to compressed `H264` or `H265` format.

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, one of the following values:
  - `nvidia` - only NVIDIA hardware encoder is supported by IFF at the moment
- `cpu_device_id`: CPU device ID
- `gpu_device_id`: GPU device ID
- `codec`: video codec to use, one of the following values:
  - `H264`
  - `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` (default)
  - for H265 codec:
    - `H265_MAIN` (default for 8-bit input)
    - `H265_MAIN10` (default for 10-bit input)
- `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` (default)
    - `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 (default)
- config_preset (default ”DEFAULT”): encoding preset, 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
    * P2
    * P3
    * P4
    * P5
    * P6
    * P7
- preset_tuning (default ”ULTRA_LOW_LATENCY”): preset tuning mode supported on desktop GPU only, one of the following modes:
  - 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 (default) - tune presets for ultra low latency streaming
- multipass (default ”DISABLED”): multi pass encoding mode. Supported on desktop GPU only. Following modes are supported:
  - DISABLED (default) - 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, one of the following:
  - on both Jetson and desktop GPU:
    * VBR - variable bit-rate mode
    * CBR (default) - 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
  execution result format:

```
{ "success": true }
```

4.3.4 `fps_limiter`

Drops frames which come faster than specified frame rate.

JSON configuration:

```
{
    "id": "fps_limit",
    "type": "fps_limiter",
    "max_processing_count": 2,
    "framerate": 0.0,
    "jitter": 0.05
}
```

parameters

- `framerate` (default 0.0): maximum output frame rate, zero or negative value means unlimited
- `jitter` (default 0.05): allowed jitter expressed in units of one period (reciprocal of `framerate`), valid range from zero to one (inclusive)

4.3.5 `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.

4.3.6 `gamma`

Applies gamma curve using LUT to `Mono` or `RGB` input images while optionally changing image bit-depth.

JSON configuration:

```
{
    "id": "oetf",
    "type": "gamma",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
    "bitdepth": 0,
    "linear": 0.0,
    "power": 1.0
}
```

formula

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

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)

parameters

- `cpu_device_id`: CPU device ID
- `bitdepth` (default 0): output bit-depth, non-positive value (e.g. default zero) keeps input bit-depth for output
- `power` (default 1.0)
- `linear` (default 0.0)

Last 2 parameters define values of corresponding variables in BT.709-like gamma formula.

references

- Recommendation ITU-R BT.709-6 (06/2015) ”Parameter values for the HDTV standards for production and international programme exchange”

4.3.7 `highlight_recovery`

Interpolates values of saturated pixels using highlight reconstruction algorithm based on ratios between Bayer channels. Input images must be in `Bayer` (unpacked) format.

JSON configuration:

```
{
    "id": "highlights",
    "type": "highlight_recovery",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
    "headroom_bits": 0,
    "number_of_interpolation_threads": 0,
    "interpolation_step": 2,
    "denoise": true,
    "rolloff": 4.0,
    "dark_rolloff": 16.0,
    "dark": 0.125,
    "threshold": 0.987
}
```

parameters

- `cpu_device_id`: CPU device ID
- `headroom_bits` (default 0): image bit-depth will be increased by this number, zero value disables processing, negative value fixes output bit-depth at 16
- `number_of_interpolation_threads` (default 0): number of processing threads, non-positive value means auto-detect (using `std::thread::hardware_concurrency()`)
- `interpolation_step` (default 2): possible values are:
  - 1 - interpolate each pixel 8 times, which may produce better results at the cost of the processing speed
  - 2 (default) - interpolate each pixel 4 times, which is faster and usually visually indistinguishable
- `denoise` (default true): whether to apply simple denoising algorithm (5x5 median filter) to the reconstructed highlights
- `rolloff` (default 4.0): force of smoothing applied to channel ratio changing over vertical and horizontal directions, use higher values to deal with fringes (e.g. due to aberrations)
- `dark_rolloff` (default 16.0): same as `rolloff`, but for dark pixels, scale together with `rolloff`
- `dark` (default 0.125): if normalized pixel value after white balance is below this value it is considered too dark and so white color is used for channel ratio calculation instead, decrease for darker scenes
- `threshold` (default 0.987): if normalized pixel value is above this value it is considered saturated and so reconstruction algorithm is applied to it

It is advised to set baseline_exposure parameter of `dng_writer` to the same value as `headroom_bits`.

4.3.8 `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) - $\mathrm{X} = \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$

All formats are stored in the memory as an array of 32-bit integers.

4.3.9 `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": 0,
    "height": 0
}
```

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.

4.3.10 `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` - called when the frame passes through the filter
  new_frame_metadata data format:

```
{
    "sequence_id": 1,
    "sequence_ts": 16832616755504369933,
    "sequence_num": 0,
    "ntp_ts": 16832616755934335837,
    "src_ts": 13738592,
    "width": 3840,
    "height": 2160,
    "offset_x": 0,
    "offset_y": 0,
    "black_level": 0,
    "exposure": 10000,
    "gain": 0.0,
    "wb_r": 1.0,
    "wb_g1": 1.0,
    "wb_g2": 1.0,
    "wb_b": 1.0,
    "wb_b_off": 0.0,
    "wb_g1_off": 0.0,
    "wb_g2_off": 0.0,
    "wb_r_off": 0.0,
    "static_metadata": {
        "ip": "127.0.0.1"
    }
}
```

  - `sequence_id`: ID of a dispatch session within which given image was dispatched, provided by source
  - `sequence_ts`: timestamp in NTP format (see RFC 5905) when current dispatch session was started
  - `sequence_num`: image sequence number
  - `ntp_ts`: image timestamp in NTP format (see RFC 5905)
  - `src_ts`: image timestamp provided by source

- `width`: image width
- `height`: image height
- `offset_x`: horizontal offset of ROI or crop position
- `offset_y`: vertical offset of ROI or crop position
- `black_level`: image black level
- `exposure`: image exposure time in microseconds
- `gain`: image gain in dB
- image white balance coefficients:
  * `wb_b_off`
  * `wb_g1`
  * `wb_g1_off`
  * `wb_g2`
  * `wb_g2_off`
  * `wb_r`
  * `wb_r_off`
- `static_metadata`: static data identical for each frame, defined in the element configuration

4.3.11 `packer`

Converts unpacked `Mono` and `Bayer` image formats into packed `Monopmsb` and `Bayerpmsb` formats compatible with DNG specification. Input images that can't be packed (e.g. with `RGB` format) are passed through as is.

JSON configuration:

```
{
    "id": "pack",
    "type": "packer",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev"
}
```

parameters

- `cpu_device_id`: CPU device ID

references

- GenICam Pixel Format Naming Convention (PFNC) Version 2.4
- TIFF 6.0 Specification
- DNG Specification (version 1.5.0.0)

4.3.12 `resizer`

Resizes the image.

JSON configuration:

```
{
    "id": "resizer",
    "type": "resizer",
    "max_processing_count": 2,
    "cpu_device_id": "cpu_dev",
    "scale": 0.0,
```

```
    "width": 1024,
    "height": 1024
}
```

parameters

- `cpu_device_id`: CPU device ID
- `scale` (default 0.0): scale factor
- `width`, `height` (optional, if `scale` is positive): dimensions of the output resized image, used if `scale` is not positive

4.3.13 `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` - called when some connection to output of this element becomes active (images begin to flow), returns empty JSON object
- `on_active_changed` - called when this element starts or stops receiving images
  on_active_changed data format:

  ```
  {
      "active": true
  }
  ```

  - `active`: whether element is currently active (is receiving images)

4.3.14 `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",
    "color": {
        "dcp_file": "color_profile.dcp",
        "temperature": 5003,
        "output_colorspace": "Custom",
        "xyz2rgb": [
```

```
            3.1338561, -1.6168667, -0.4906146,
           -0.9787684,  1.9161415,  0.0334540,
            0.0719453, -0.2289914,  1.4052427
        ]
    },
    "switch_red_and_blue": false,
    "proc_num_threads": 0
}
```

parameters

- `cpu_device_id`: CPU device ID
- `custom_params` (optional): array of custom parameters from xiAPI
- `image_format` (default "RGB32"): output xiAPI image data format, one of the following:
  - `MONO8`
  - `MONO16`
  - `RAW8`
  - `RAW16`
  - `RGB24`
  - `RGB32` (default)
  - `RGB48`
  - `RGB64`
- `color` (optional):
  - `dcp_file` (required, if `color` section is present): path to DNG color profile file (only color matrices are used from it, `ForwardMatrix1` tag is required to be present)
  - `temperature` (default 5003): white balance temperature, used for color matrix interpolation in case of dual-illuminant color profiles
  - `output_colorspace` (default "Custom"): output color space, used for color matrix calculation (gamma is not affected), one of the following:
    - `Custom` (default) - custom color space as specified by `xyz2rgb` setting (see below)
    - `ACES`
    - `ACEScg`
    - `DisplayP3`
    - `ProPhotoRGB`
    - `Rec709` - same as sRGB
    - `Rec2020`
  - `xyz2rgb` (default XYZ D50 to sRGB matrix): 3x3 XYZ D50 to RGB matrix of floats, which defines `Custom` output color space
- `switch_red_and_blue` (default false): whether to switch to RGB output channel order instead of xiAPI default BGR, will automatically adjust color matrix settings as required
- `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.

4.3.15 `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",
    "color": {
        "dcp_file": "color_profile.dcp",
        "temperature": 5003,
        "xyz2rgb": [
             3.1338561, -1.6168667, -0.4906146,
            -0.9787684,  1.9161415,  0.0334540,
             0.0719453, -0.2289914,  1.4052427
        ]
    },
    "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" },
        { "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
- `color` (optional): common color correction settings, can be modified at runtime:
  - `dcp_file` (optional): path to DNG color profile file (`ForwardMatrix1` tag is required to be present)
  - `temperature` (default 5003): white balance temperature, used for color matrix and LUT interpolation in case of dual-illuminant color profiles
  - `xyz2rgb` (default XYZ D50 to sRGB matrix): 3x3 XYZ D50 to RGB matrix of floats, which defines `Custom` output color space
- `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

Import adapters

Exactly one import adapter must exist in the `cuda_processor` pipeline and it must be the first element (must be used in `connections` section at least once as `src` and never as `dst`).

`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"
}
```

`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"
}
```

Export adapters

Export adapters must be the last elements in the `cuda_processor` pipeline (each adapter must be used in `connections` section exactly once as `dst` and never as `src`).

Common required parameter for export adapters is:

```
{
    "output_name": "out"
}
```

- `output_name`: name of the `cuda_processor` element output for this export adapter (use "out" for default output)

`export_to_device`

Copies data to CUDA device buffer, optionally converting to specified format.

JSON configuration:

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

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)

w

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

(h)

$w/2$

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

($\frac{h}{2}$)

$w/2$

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

($\frac{h}{2}$)

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

$2 \cdot w$

h

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

w

$\frac{h}{2}$

LSB(U_{00})	MSB(U_{00})	LSB(U_{10})	MSB(U_{10})	...
LSB(U_{01})	MSB(U_{01})	LSB(U_{11})	MSB(U_{11})	...
...	...	...	...	...

w

$\frac{h}{2}$

LSB(V_{00})	MSB(V_{00})	LSB(V_{10})	MSB(V_{10})	...
LSB(V_{01})	MSB(V_{01})	LSB(V_{11})	MSB(V_{11})	...
...	...	...	...	...

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

w

h

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

w

$\frac{h}{2}$

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

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

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

$2 \cdot w$

h

LSB($\hat{Y}_{00}$)	MSB($\hat{Y}_{00}$)	LSB($\hat{Y}_{10}$)	MSB($\hat{Y}_{10}$)	...
LSB($\hat{Y}_{01}$)	MSB($\hat{Y}_{01}$)	LSB($\hat{Y}_{11}$)	MSB($\hat{Y}_{11}$)	...
...	...	...	...	...

$2 \cdot w$

$\frac{h}{2}$

LSB($\hat{U}_{00}$)	MSB($\hat{U}_{00}$)	LSB($\hat{V}_{00}$)	MSB($\hat{V}_{00}$)	LSB($\hat{U}_{10}$)	MSB($\hat{U}_{10}$)	LSB($\hat{V}_{10}$)	MSB($\hat{V}_{10}$)	...
LSB($\hat{U}_{01}$)	MSB($\hat{U}_{01}$)	LSB($\hat{V}_{01}$)	MSB($\hat{V}_{01}$)	LSB($\hat{U}_{11}$)	MSB($\hat{U}_{11}$)	LSB($\hat{V}_{11}$)	MSB($\hat{V}_{11}$)	...
...	...	...	...	...	...	...	...	...

parameters

- `output_format`: output format, one of the following:
  - with conversion (rows are aligned to 4 byte boundaries):
    - `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
  - without conversion:
    * `Mono8` - Monochrome 8-bit
    * `Mono12` - Monochrome 12-bit unpacked
    * `Mono16` - Monochrome 16-bit
    * `BayerRG8` - Bayer Red-Green 8-bit
    * `BayerRG12` - Bayer Red-Green 12-bit unpacked
    * `BayerRG16` - Bayer Red-Green 16-bit
    * `BayerBG8` - Bayer Blue-Green 8-bit
    * `BayerBG12` - Bayer Blue-Green 12-bit unpacked
    * `BayerBG16` - Bayer Blue-Green 16-bit
    * `BayerGR8` - Bayer Green-Red 8-bit
    * `BayerGR12` - Bayer Green-Red 12-bit unpacked
    * `BayerGR16` - Bayer Green-Red 16-bit
    * `BayerGB8` - Bayer Green-Blue 8-bit
    * `BayerGB12` - Bayer Green-Blue 12-bit unpacked
    * `BayerGB16` - Bayer Green-Blue 16-bit
    * `RGB8` - Red-Green-Blue 8-bit
    * `RGB12` - Red-Green-Blue 12-bit unpacked
    * `RGB16` - Red-Green-Blue 16-bit

For format description see also GenICam Pixel Format Naming Convention (PFNC) Version 2.4.

`export_to_host`

Copies data to CPU buffer, optionally converting to specified format.

JSON configuration:

```
{
    "id": "export_to_host",
    "type": "export_to_host",
    "output_name": "out",
    "output_format": "RGB16"
}
```

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_name": "out",
    "output_format": "Histogram3Bayer256Int"
}
```

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

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

(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`: output format, one of the following, where `<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

Image filters must be intermediate elements in the `cuda_processor` pipeline (each filter must be used in `connections` section exactly once as `dst` and at least once as `src`).

`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

---

`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

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

`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",
    "from": "Camera",
    "to": "Custom",
    "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

- `from` (default ”Camera”): input color space, see description of `to` parameter below for possible values
- `to` (default ”Custom”): output color space, one of the following:
  - `Camera` (default for `from`) - camera color space as specified in global `cuda_processor` color/dcp_file setting (valid only for `from` parameter)
  - `Custom` (default for `to`) - custom color space as specified in global `cuda_processor` color/xyz2rgb setting (valid only for `to` parameter)
  - `ACES`
  - `ACEScg`
  - `DisplayP3`
  - `ProPhotoRGB`
  - `Rec709` - same as sRGB
  - `Rec2020`
- `matrix` (optional): color correction matrix M in row scan order, if present overrides `from` and `to` parameters

If `from` parameter is set to default ”Camera” value, but DNG color profile is not specified, then color correction matrix defaults to identity matrix (which still can be overridden by `matrix` parameter).

`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.

---

demosaic

Transforms raw Bayer image into RGB image.

JSON configuration:

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

parameters

- `algorithm`: algorithm to use, one of the following:
  - `HQLI` - High Quality Linear Interpolation, window 5x5, avg. PSNR ~36 dB for Kodak data set
  - `L7` - High Quality Linear Interpolation, window 7x7, 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 11x11, avg. PSNR ~39 dB for Kodak data set
  - `MG` - Multiple Gradients, window 23x23, avg. PSNR ~40.5 dB for Kodak data set, doesn't support 8-bit input

---

denoise and raw_denoise

Removes noise from the image using Discrete Wavelet Transform (DWT) and thresholding. RGB images are split to Y, Cb and Cr channels for processing. Bayer images are processed as one channel, but each color plane (R, G1, G2 and B) separately.

JSON configuration for RGB images:

```
{
    "id": "denoise",
    "type": "denoise",
    "wavelet_type": "CDF53",
    "dwt_levels": 4,
    "threshold_function": "GARROTE",
    "threshold": [ 0.0, 0.0, 0.0 ],
    "threshold_per_level": [ [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ],
                             [ 1.0, 1.0, 1.0 ] ]
}
```

JSON configuration for Mono and Bayer images:

```
{
    "id": "denoise",
    "type": "raw_denoise",
    "wavelet_type": "CDF53",
    "dwt_levels": 4,
    "threshold_function": "GARROTE",
    "threshold": 0.0,
    "threshold_per_level": [ 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 ↩
        ]
}
```

parameters

- `type`: one of the following, depending on the input image format:
  - `denoise` - use for Mono and RGB images
  - `raw_denoise` - use for Bayer images
- `wavelet_type` (default "CDF53"): wavelet type, one of the following:
  - `CDF53` (default)
  - `CDF97`
- `dwt_levels` (default 4): number of DWT levels (from 1 to 11)
- `threshold_function` (default "GARROTE"): threshold function, one of the following:
  - `GARROTE` (default)
  - `HARD`
  - `SOFT`
- `threshold` (default 0.0): thresholds for each channel (by default image is not modified)
- `threshold_per_level` (default 1.0): threshold factors per wavelet level (in ascending order) for each channel (by default `threshold` is used as is for all levels)

`exposure_indicator`

Highlights under- and over-exposed image areas with blinking effect.

JSON configuration:

```
{
    "id": "exposure_indicator",
    "type": "exposure_indicator",
    "underexposure": 0.01,
    "middlegray": 0.18,
    "overexposure": 0.99,
    "halfperiod": 10
}
```

formula

$$out = \begin{cases} white\_level \cdot \mathrm{T}\left(\dfrac{in}{white\_level}\right) & n \bmod (2 \cdot halfperiod) < halfperiod \\ in & n \bmod (2 \cdot halfperiod) \geq halfperiod \end{cases}$$

where n is a zero-based sequence number of the current image

$$\mathrm{T}(x) = \begin{cases} middlegray & x \leq underexposure \\ x & underexposure < x < overexposure \\ middlegray & x \geq overexposure \end{cases}$$

parameters

- `underexposure` (default 0.01): maximum normalized (from 0 to 1) pixel value for it to be considered under-exposed
- `middlegray` (default 0.18): normalized (from 0 to 1) pixel value to use for highlighting under- and over-exposed areas
- `overexposure` (default 0.99): minimum normalized (from 0 to 1) pixel value for it to be considered over-exposed
- `halfperiod` (default 10): how many images to process before switching between highlight and pass-through modes

---

`ffc`

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

JSON configuration:

```
{
    "id": "ffc",
    "type": "ffc",
    "dark_field": "darkfield-12.raw",
    "flat_field": "flatfield-12.raw",
    "bitdepth": 12,
    "width": 1024,
    "offset_x": 0,
    "offset_y": 0
}
```

formula

$$D_{xy} = dark_{xy} \cdot 2^{in\_bitdepth - bitdepth}$$

$$F_{xy} = flat_{xy} \cdot 2^{in\_bitdepth - bitdepth}$$

$$G_{xy} = \frac{white\_level}{white\_level - \overline{D_{bayer}}} \cdot \frac{\overline{(F-D)_{bayer}}}{F_{xy} - D_{xy}}$$

$$out_{xy} = (in_{xy} - D_{xy}) \cdot \begin{cases} \frac{1}{8} & G_{xy} < \frac{1}{8} \\ G_{xy} & \frac{1}{8} \leq G_{xy} \leq 8 \\ 8 & G_{xy} > 8 \end{cases}$$

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.

---

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` (default "gamma"): function describing the applied curve, one of the following:
  - `gamma` (default) - BT.709-like gamma function
  - `hlg` - Hybrid Log-Gamma function
- `linear` (default 0.0)
- `power` (default 1.0)

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`.

`huesatmap`

Applies 3D HSV LUT to the RGB image.

JSON configuration:

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

Application algorithm is described in DNG Specification (version 1.5.0.0), end of chapter 6 (page 88). LUT data is taken from DNG color profile specified in global `cuda_processor` color settings. Input data has to be linear RGB in ProPhotoRGB color space for correct results.

`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.

`undistort`

Corrects lens distortion using remapping operation.

JSON configuration:

```
{
    "id": "undistort",
    "type": "undistort",
    "lens_profile": "lens_profile.json",
    "interpolator": "linear"
}
```

Lens profile file format:

```
{
    "distortion": {
        "type": "radial_simple",
        "centerX": 1920.0,
        "centerY": 1080.0,
        "coefficients": [ 1.0 ]
    }
}
```

formula

$$r = \sqrt{(x_{in} - centerX)^2 + (y_{in} - centerY)^2}$$

$$x_{out} = centerX + \mathsf{L}(r) \cdot (x_{in} - centerX)$$

$$y_{out} = centerY + \mathsf{L}(r) \cdot (y_{in} - centerY)$$

polynomial radial distortion model

$$\mathsf{L}(r) = K_0 + K_1 \cdot r + K_2 \cdot r^2 + \cdots$$

parameters

- `lens_profile`: path to lens profile file
- `interpolator` (default "linear"): used interpolation algorithm, one of the following:
  - `linear` (default) - linear interpolation
  - `cubic` - cubic interpolation
  - `cubic2p_bspline` - two-parameter cubic filter (B=1, C=0)
  - `cubic2p_catmullrom` - two-parameter cubic filter (B=0, C=1/2)
  - `cubic2p_b05c03` - two-parameter cubic filter (B=1/2, C=3/10)
  - `super` - super sampling
  - `lanczos` - Lanczos filtering

lens profile parameters

- `type`: lens distortion model, one of the following:
  - `radial_simple` - polynomial radial distortion model
- `centerX` (optional): horizontal coordinate of distortion center in pixels, center of the image by default
- `centerY` (optional): vertical coordinate of distortion center in pixels, center of the image by default
- `coefficients`: coefficients for the distortion model formula ($K_0, K_1, \ldots$)

All output pixels mapped outside the input image are filled with black color.

`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

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 $\mathrm{G2}, \Upsilon(3) = \mathrm{B}$

histogram stretch algorithm

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

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

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

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

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

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"): algorithm to use, one of the following:
  - `simple` (default) - 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).

5 IFF SDK C 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* title, const char* message);
```

Add 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).
`title`	Message title to be shown after level in square brackets (i.e. `[level] [title] message`).
`message`	Message to be logged.

5.1.4 iff_create_chain()

```
iff_chain_handle_t iff_create_chain(const char* config, iff_error_handler_t  ↩
    error_handler, void* private_data);
```

Create a new IFF processing chain according to passed configuration.

Parameters:

`config`	Configuration of IFF chain to create in JSON format. See Chain description format.
`error_handler`	Pointer to a function that is called if error occurred during processing chain lifetime. See `iff_error_handler_t`.
`private_data`	Pointer to the user data. This pointer will be passed as parameter to `on_error` function with each invocation.

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_callback_t result_handler, void* private_data);
```

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.
`result_handler`	Pointer to a function that is called by SDK to return values of requested elements parameters. See `iff_callback_t`.
`private_data`	Pointer to the user data. This pointer will be passed as parameter to `result_handler` function with each invocation.

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,  ↩
   iff_callback_t result_handler, void* private_data);
```

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.
result_handler	Pointer to a function that is called by SDK to return the command execution result. See iff_callback_t.
private_data	Pointer to the user data. This pointer will be passed as parameter to result_handler function with each invocation.

5.1.9 iff_set_callback()

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

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.
private_data	Pointer to the user data. This pointer will be passed as parameter to callback function with each invocation.

5.1.10 iff_set_export_callback()

```
void iff_set_export_callback(iff_chain_handle_t chain_handle, const char*  ↩
   exporter_id, iff_image_handler_t image_handler, 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.
image_handler	Pointer to export callback function. See iff_image_handler_t.
private_data	Pointer to the user data. This pointer will be passed as parameter to export_func function with each invocation.

5.1.11 iff_get_import_buffer()

```
void* iff_get_import_buffer(iff_chain_handle_t chain_handle, const char*  ↩
   importer_id, size_t* size);
```

Get the first available buffer of the specified importer element (see frame_importer). Buffer can be used to pass the frame data from user code to IFF SDK library.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function.
importer_id	ID of the importer element. See frame_importer.
size	Pointer to a variable to store the size of the returned buffer.

Returns:
Pointer to available import buffer. If no buffers are available returns nullptr.

5.1.12 iff_push_import_buffer()

```
bool iff_push_import_buffer(iff_chain_handle_t chain_handle, const char*  ↩
   importer_id, void* buffer, iff_image_metadata metadata);
```

Push filled import buffer (see iff_get_import_buffer()) to the importer element (see frame_importer) for dispatching to the processing pipeline.

Parameters:

chain_handle	Handle of the processing chain, returned by iff_create_chain() function.
importer_id	ID of the importer element. See frame_importer.
buffer	Import buffer pointer. See iff_get_import_buffer().
metadata	Corresponding image metadata, that will be passed further into the processing pipeline. See iff_image_metadata. If generate_timestamps setting of frame_importer element is set to false a valid src_ts field value that increases monotonically from frame to frame must be specified.

Note
Values of `height`, `width` and `padding` fields of `metadata` parameter are ignored and replaced with the values specified in the importer element configuration. See `frame_importer`.

Returns:
`true` if buffer was pushed successfully, `false` if an error occurred.

5.1.13 `iff_release_buffer()`

```
bool iff_release_buffer(iff_chain_handle_t chain_handle, const char* element_id,  ↩
   void* buffer);
```

Release buffer previously obtained from `frame_importer` element by calling `iff_get_import_buffer()` or from `frame_exporter` element via export callback (see `iff_set_export_callback()`), if it is used in asynchronous mode.

Parameters:

`chain_handle`	Handle of the processing chain, returned by `iff_create_chain()` function.
`element_id`	The ID of the element that the buffer belongs to.
`buffer`	Pointer to the buffer to be released.

Returns:
`true` if buffer was released successfully, `false` if an error occurred.

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
{
    uint32_t width;
    uint32_t height;
    uint32_t offset_x;
```

```
    uint32_t offset_y;

    uint64_t src_ts;
    uint64_t ntp_ts;

    uint32_t exposure;
    uint32_t black_level;
    float gain;

    iff_wb_params wb;

    unsigned char sequence_id;

    size_t   padding;
} iff_image_metadata;
```

Members:

`width`	Image width in pixels.
`height`	Image height in pixels.
`offset_x`	Horizontal offset of ROI or crop position.
`offset_y`	Vertical offset of ROI or crop position.
`src_ts`	Image timestamp provided by source.
`ntp_ts`	Image timestamp in NTP format (see RFC 5905).
`exposure`	Image exposure time in microseconds.
`black_level`	Image black level.
`gain`	Image gain in dB.
`wb`	Image white balance coefficients.
`sequence_id`	ID of a dispatch session within which given image was dispatched provided by source.
`padding`	Image padding in bytes.

5.3 Types

5.3.1 `iff_error_handler_t`

```
typedef void(*iff_error_handler_t)(const char* element_id, int error_code, void* ↩
    private_data);
```

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_id`	ID of the chain element that triggered the error.
`error_code`	Code of an error.
`private_data`	Pointer to the user data that was passed to `iff_create_chain()` call.

5.3.2 iff_callback_t

```
typedef void(*iff_callback_t)(const char* data, void* private_data);
```

Function pointer of this type must be passed as a parameter to the `iff_get_params()`, `iff_execute()` and `iff_set_callback()` calls. IFF will call the function at the given pointer to return a JSON string containing the execution result of the corresponding function or callback data if it was set as an element callback. This JSON string will be passed to the function as a parameter.

Parameters:

`data`	Returned data in JSON format.
`private_data`	Pointer to the user data that was passed to `iff_get_params()`, `iff_execute()` or `iff_set_callback()` call.

For `iff_get_params()` function format of the output JSON string is the same as format of input JSON string passed to `iff_set_params()` function. See Set parameters input format.

For `iff_execute()` function format of the output JSON string is specified in the component description. Most commonly it's `{ "success": true }`.

5.3.3 iff_image_handler_t

```
typedef void(*iff_image_handler_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 user code across IFF SDK library boundaries.

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, unless `async_mode` parameter in corresponding `frame_exporter` element is set to `true`.
`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.

5.3.4 iff_error_code

```
typedef enum iff_error_code
{
    camera_disconnected     = 0x4001,
    end_of_stream           = 0x4002,

    element_not_found       = 0x4003,
    element_not_ready       = 0x4004,
```

```
    image_queue_overflow    = 0x4005,
    invalid_input_data      = 0x4006,
    out_of_memory           = 0x4007,
    processing_failed       = 0x4008,

    parameter_not_found     = 0x4009,
    invalid_parameter_value = 0x400a,
    command_not_found       = 0x400b,

    camera_reset            = 0x400c
} iff_error_code;
```

IFF SDK error codes enumeration. See `iff_error_handler_t` for more information on error handling.

6 IFF SDK library C++ wrapper

IFF SDK also provides C++ wrapper around C library interface. Wrapper is implemented as a single C++ header file **iffwrapper.hpp** located in the **include** directory of IFF SDK package. It uses `iffwrapper` namespace.

6.1 Enumerations

6.1.1 log_level

```
enum class log_level
{
    debug,
    warning,
    error,
    info
};
```

Message severity level.

6.2 Type aliases

6.2.1 image_metadata

```
using wb_params = iff_wb_params;
using image_metadata = iff_image_metadata;
```

See iff_image_metadata.

6.2.2 error_code

```
using error_code = iff_error_code;
```

See iff_error_code.

6.2.3 error_handler_t

```
using error_handler_t = std::function<void(const std::string& element_id, int  ↩
    error_code)>;
```

Function object of this type must be passed to chain() constructor when creating a new chain. IFF will call the function stored in this object whenever an error occurs while chain is processing the image or executing a user request.

Parameters:

element_id	ID of the chain element that triggered the error.
error_code	Code of an error.

6.2.4 callback_t

```
using callback_t = std::function<void(const std::string& data)>;
```

Function object of this type must be passed as a parameter to the get_params(), execute() and set_callback() calls. IFF will call the function stored in this object to return a JSON string containing the execution result of the corresponding function or callback data if it was set as an element callback. This JSON string will be passed to the function as a parameter.

Parameters:

data	Returned data in JSON format.

For get_params() function format of the output JSON string is the same as format of input JSON string passed to set_params() function. See Set parameters input format.

For execute() function format of the output JSON string is specified in the component description. Most commonly it's { "success": true }.

6.2.5 image_handler_t

```
using image_handler_t = std::function<void(const void* data, size_t size,  ↩
    image_metadata metadata)>;
```

Function object of this type must be passed to set_export_callback() function call. The function stored in this object is called by exporter element when a new frame is received to send it to the user code across IFF SDK library boundaries.

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, unless async_mode parameter in corresponding frame_exporter element is set to true.
size	Size of image data in bytes.
metadata	Pointer to the image metadata structure. See image_metadata.

6.3 Functions

6.3.1 initialize()

```
void initialize(const std::string& 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 finalize() function. If an instance of IFF framework is already initialized, parameter config is ignored.

Parameters:

`config`	Configuration of IFF framework in JSON format.

6.3.2 `finalize()`

```
void 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.

6.3.3 `log()`

```
void log(log_level level, const std::string& title, const std::string& message);
```

Add 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: `log_level::debug`, `log_level::warning`, `log_level::error`, `log_level::info` (always logged). See log_level enumeration.
`title`	Message title to be shown after level in square brackets (i.e. `[level] [title] message`).
`message`	Message to be logged.

6.4 Classes

6.4.1 `chain`

Represents an IFF processing chain and simplifies corresponding routines.

Constructor & Destructor

`chain()`

```
chain(const std::string& config, error_handler_t error_handler);
// copy is forbidden
chain(const chain&) = delete;
chain& operator=(const chain&) = delete;
// move is allowed
chain(chain&&) noexcept;
chain& operator=(chain&&) noexcept;
```

Create a new IFF processing chain according to passed configuration.

Parameters:

`config`	Configuration of IFF chain to create in JSON format. See Chain description format.
`error_handler`	Function that is called if error occurred during processing chain lifetime. See `error_handler_t`.

~chain()

```
~chain();
```

Finalize processing chain and release all its resources.

Public Member Functions

execute()

```
void execute(const std::string& command, callback_t result_handler = [](const std ↩
    ::string&){});
```

Request execution of the specified command from the chain element.

Parameters:

`command`	Command to execute and its parameters if any in JSON format. See Execute input format.
`result_handler`	Function that is called by SDK to return the command execution result. Can be omitted. See `callback_t`.

get_params()

```
void get_params(const std::string& params, callback_t result_handler);
```

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

Parameters:

`params`	Elements parameters names to get in JSON format. See Get parameters input format.
`result_handler`	Function that is called by SDK to return values of requested elements parameters. See `callback_t`.

set_params()

```
void set_params(const std::string& params);
```

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

Parameters:

params	Chain elements parameters and its values to set. See Set parameters input format.

set_callback()

```
bool set_callback(const std::string& name, callback_t callback);
```

Set the given function to the specified element callback.

Parameters:

name	Element callback name in the format **<element ID>/<callback name>**.
callback	Callback function. See callback_t.

Returns:
true unless callback is not set.

set_export_callback()

```
bool set_export_callback(const std::string& exporter_id, image_handler_t ↩
    image_handler);
```

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:

exporter_id	ID of the exporter element. See frame_exporter.
image_handler	Export callback function. See image_handler_t.

Returns:
true unless callback is not set.

get_import_buffer()

```
void* get_import_buffer(const std::string& importer_id, size_t* size);
```

Get the first available buffer of the specified importer element (see frame_importer). Buffer can be used to pass the frame data from user code to IFF SDK library.

Parameters:

importer_id	ID of the importer element. See frame_importer.
size	Pointer to a variable to store the size of the returned buffer.

Returns:
Pointer to available import buffer. If no buffers are available, returns `nullptr`.

`push_import_buffer()`

```
bool push_import_buffer(const std::string& importer_id, void* buffer,  ↩
    image_metadata metadata);
```

Push filled import buffer (see `get_import_buffer()`) to the importer element (see `frame_importer`) for dispatching to the processing pipeline.

Parameters:

`importer_id`	ID of the importer element. See `frame_importer`.
`buffer`	Import buffer pointer. See `get_import_buffer()`.
`metadata`	Corresponding image metadata, that will be passed further into the processing pipeline. See `iff_image_metadata`. If `generate_timestamps` setting of `frame_importer` element is set to `false` a valid `src_ts` field value that increases monotonically from frame to frame must be specified.

Note
Values of `height`, `width` and `padding` fields of `metadata` parameter are ignored and replaced with the values specified in the importer element configuration. See `frame_importer`.

Returns:
`true` if buffer was pushed successfully, `false` if an error occurred.

`release_buffer()`

```
bool release_buffer(const std::string& element_id, void* buffer);
```

Release buffer previously obtained from `frame_importer` element by calling `get_import_buffer()` or from `frame_exporter` element via export callback (see `set_export_callback()`) if it is used in asynchronous mode.

Parameters:

`element_id`	The ID of the element that the buffer belongs to.
`buffer`	Pointer to the buffer to be released.

Returns:
`true` if buffer was released successfully, `false` if an error occurred.

7 IFF SDK Python bindings

In IFF SDK, all algorithms are implemented in C++. But these algorithms can be used from Python. This is made possible by bindings that create a bridge between C++ and Python, allowing users to call C++ functions from Python. These bindings are provided by **`iffsdkpy`** shared library module. You can easily use them by simply importing this module from Python (assuming **`iffsdkpy`** module is located where Python interpreter is able to find it):

```
import iffsdkpy
```

7.1 Enumerations

7.1.1 `log_level`

Message severity level. See `log_level`.

7.1.2 `error_code`

IFF SDK error codes enumeration. See `iff_error_code`.

7.2 Functions

7.2.1 `initialize()`

```
initialize(config: str) -> None
```

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 `finalize()` function. If an instance of IFF framework is already initialized, parameter `config` is ignored.

Parameters:

`config`	Configuration of IFF framework in JSON format.

7.2.2 `finalize()`

```
finalize() -> None
```

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.

7.2.3 `log()`

```
log(level: log_level, title: str, message: str) -> None
```

Add 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: `log_level.debug`, `log_level.warning`, `log_level.error`, `log_level.info` (always logged). See log_level enumeration.
`title`	Message title to be shown after level in square brackets (i.e. `[level] [title] message`).
`message`	Message to be logged.

7.3 Classes

7.3.1 `wb_params`

Image white balance coefficients. See `iff_image_metadata`.

7.3.2 `image_metadata`

Image metadata structure contains parameters of a specific processed image. See `iff_image_metadata`.

7.3.3 `Chain`

Represents an IFF processing chain.

Constructor & Destructor

`Chain()`

```
Chain(config: str, error_handler: Callable[[str, int], None]) -> Chain
```

Create a new IFF processing chain according to passed configuration.

Parameters:

`config`	Configuration of IFF chain to create in JSON format. See Chain description format.
`error_handler`	Function that is called if error occurred during processing chain lifetime. See `error_handler_t`.

Public Member Functions

`execute()`

```
execute(command: str, result_handler: Optional[Callable[[str], None]]) -> None
```

Request execution of the specified command from the chain element.

Parameters:

`command`	Command to execute and its parameters if any in JSON format. See Execute input format.
`result_handler`	Function that is called by SDK to return the command execution result. Can be omitted. See `callback_t`.

`get_params()`

```
get_params(params: str, result_handler: Callable[[str], None]) -> None
```

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

Parameters:

`params`	Elements parameters names to get in JSON format. See Get parameters input format.
`result_handler`	Function that is called by SDK to return values of requested elements parameters. See `callback_t`.

`set_params()`

```
set_params(params: str) -> None
```

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

Parameters:

`params`	Chain elements parameters and its values to set. See Set parameters input format.

`set_callback()`

```
set_callback(name: str, callback: Callable[[str], None]) -> bool
```

Set the given function to the specified element callback.

Parameters:

`name`	Element callback name in the format **<element ID>/<callback name>**.
`callback`	Callback function. See `callback_t`.

Returns:

`True` unless callback is not set.

`set_export_callback()`

```
set_export_callback(exporter_id: str, image_handler: Callable[[memoryview,  ↩
    image_metadata], None]) -> bool
```

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:

exporter_id	ID of the exporter element. See frame_exporter.
image_handler	Export callback function. See image_handler_t (size parameter is missing in Python callback, since it is available through nbytes attribute of memoryview object passed as data callback parameter).

Returns:
True unless callback is not set.

get_import_buffer()

```
get_import_buffer(importer_id: str) -> memoryview
```

Get the first available buffer of the specified importer element (see frame_importer). Buffer can be used to pass the frame data from user code to IFF SDK library.

Parameters:

importer_id	ID of the importer element. See frame_importer.

Returns:
memoryview object that references an import buffer. If no buffers are available, nbytes attribute of returned object equals zero.

push_import_buffer()

```
push_import_buffer(importer_id: str, buffer: memoryview, metadata: image_metadata) ↩
    -> bool
```

Push filled import buffer (see get_import_buffer()) to the importer element (see frame_importer) for dispatching to the processing pipeline.

Parameters:

importer_id	ID of the importer element. See frame_importer.
buffer	memoryview object that references an import buffer. See get_import_buffer().
metadata	Corresponding image metadata, that will be passed further into the processing pipeline. See iff_image_metadata. If generate_timestamps setting of frame_importer element is set to false a valid src_ts field value that increases monotonically from frame to frame must be specified.

Note
Values of `height`, `width` and `padding` fields of `metadata` parameter are ignored and replaced with the values specified in the importer element configuration. See `frame_importer`.

Returns:
`True` if buffer was pushed successfully, `False` if an error occurred.

`release_buffer()`

```
release_buffer(element_id: str, buffer: memoryview) -> bool
```

Release buffer previously obtained from `frame_importer` element by calling `get_import_buffer()` or from `frame_exporter` element via export callback (see `set_export_callback()`) if it is used in asynchronous mode.

Parameters:

`element_id`	The ID of the element that the buffer belongs to.
`buffer`	`memoryview` object that references the buffer to be released.

Returns:
`True` if buffer was released successfully, `False` if an error occurred.

8 IFF SDK configuration

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

8.1 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.

8.2 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` (default)
  - `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>/<chain 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, one of the following:
  - `cpu`
  - `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

8.3 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`

8.3.1 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.

8.3.2 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:

- `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` (default)
  - `weak`

8.3.3 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:

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

8.3.4 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:

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

8.4 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()`.

8.4.1 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.

8.4.2 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.

8.4.3 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.

8.5 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:

- `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.

8.5.1 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 `main`.

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 `main`.

9 Sample applications

9.1 `farsight`

`farsight` is the most basic and general sample application for C API of MRTech IFF SDK. It is located in `samples/01_streaming` directory of IFF SDK package. Application 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

9.2 `farsightcpp`

`farsightcpp` is the most basic and general sample application for C++ API of MRTech IFF SDK. It is located in `samples/01_streaming_cpp` directory of IFF SDK package. Application comes with example configuration file (`farsightcpp.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

9.3 farsightpy

`farsight.py` is the most basic and general sample application for Python API of MRTech IFF SDK. It is located in `samples/01_streaming_py` directory of IFF SDK package. Application 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

9.4 imagebroker

`imagebroker` application demonstrates how to export images to the user code from C API of MRTech IFF SDK. It is located in `samples/02_export` directory of IFF SDK package. Application 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
  - gamma
  - image format conversion
- automatic control of exposure time and white balance
- image export to the user 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).

9.5 imagebrokercpp

`imagebrokercpp` application demonstrates how to export images to the user code from C++ API of MRTech IFF SDK. It is located in `samples/02_export_cpp` directory of IFF SDK package. Application comes with example configuration file (`imagebrokercpp.json`) providing the following functionality:

- acquisition from XIMEA camera
- 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
- image export to the user 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).

9.6 imagebrokerpy

`imagebroker.py` application demonstrates how to export images to the user code from Python API of MRTech IFF SDK. It is located in `samples/02_export_py` directory of IFF SDK package. Application 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
  - gamma
  - image format conversion
- automatic control of exposure time and white balance
- image export to the user 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).

9.7 crowsnest

Web interface sample called `crowsnest` demonstrates the possibility to control runtime parameters of MRTech 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.

9.7.1 Installation

`linux/install.sh` installation script is provided as a reference. It was tested on Ubuntu 20.04, 24.04, NVIDIA Jetson Linux (L4T) 32.7, 35.4, 36.2 and 36.4.3. On success it prints out instructions for final setup steps.

9.7.2 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/
```

9.8 spectraprofiler

`spectraprofiler` application implements a workflow to create DNG color profiles (DCP), that can be used together with MRTech IFF SDK. It shares most of the C++ code with `imagebroker` example IFF SDK application, but also includes `coloric.py` Python script for visual color target grid positioning and uses dcamprof and Argyll CMS for DCP file generation. Application is located in `samples/04_color` directory of IFF SDK package. It comes with example configuration files (`spectraprofiler.json` and `res/coloric.json`) suited for XIMEA cameras and standard 24-patch color reference target (e.g. Calibrite ColorChecker Passport Photo 2). See `linux` and `windows` directories for helper scripts to install required dependencies (e.g. OpenCV library). Operation is controlled using a keyboard:

- 1 decreases exposure
- 2 increases exposure
- `Tab` captures an image and starts the profile generation procedure (further instructions are shown on the screen)

9.9 streamadapter

`streamadapter` application demonstrates how to export images to the GStreamer pipeline from MRTech IFF SDK. It is located in `samples/05_gstreamer` directory of IFF SDK package. Application comes with example configuration file (`streamadapter.json`) providing the following functionality:

- acquisition from XIMEA camera

- 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
- image export to the user code

9.9.1 Usage

GStreamer pipeline description is passed through command line arguments, with `streamadapter` acting as a source. For example, each incoming image can be dumped to a separate file with this command:

```
./streamadapter ! multifilesink
```

9.10 lensprofiler

`lensprofiler` application implements a workflow to create lens profiles, that can be used together with MRTech IFF SDK. It shares most of the C++ code with `imagebroker` example IFF SDK application, but also includes `ldc.py` Python script for chessboard target auto-detection and radial lens distortion model estimation, which is based on the source code provided with Algebraic Lens Distortion Model Estimation article by Luis Alvarez, Luis Gomez, and J. Rafael Sendra published in Image Processing On Line journal. Application is located in `samples/06_lens` directory of IFF SDK package. It comes with example configuration files (`lensprofiler.json` and `res/ldc.json`) suited for XIMEA cameras and standard 10x7 chessboard target (such as included `res/chessboard.png`). See `linux` and `windows` directories for helper scripts to install required dependencies (e.g. OpenCV library). Operation is controlled using a keyboard:

- 1 decreases exposure
- 2 increases exposure
- `Tab` captures an image and starts the profile generation procedure (further instructions are shown on the screen)

9.11 imagefilter

`imagefilter` application demonstrates how to implement a custom image filter (crosshair overlay) using C API of MRTech IFF SDK. It is located in `samples/07_filter` directory of IFF SDK package. Application comes with example configuration file (`imagefilter.json`) demonstrating the following functionality:

- acquisition from XIMEA camera
- 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
- image export to the user code
- image import from the user code
- H.264 encoding
- RTSP streaming
- HTTP control interface

9.12 `imagefiltercpp`

`imagefiltercpp` application demonstrates how to implement a custom image filter (crosshair overlay) using C++ API of MRTech IFF SDK. It is located in `samples/07_filter_cpp` directory of IFF SDK package. Application comes with example configuration file (`imagefiltercpp.json`) demonstrating the following functionality:

- acquisition from XIMEA camera
- 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
- image export to the user code
- image import from the user code
- H.264 encoding
- RTSP streaming
- HTTP control interface

9.13 imagefilterpy

`imagefilter.py` application demonstrates how to implement a custom image filter (crosshair overlay) using Python API of MRTech IFF SDK. It is located in `samples/07_filter_py` directory of IFF SDK package. Application comes with example configuration file (`imagefilter.json`) demonstrating the following functionality:

- acquisition from XIMEA camera
- 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
- image export to the user code
- image import from the user code
- H.264 encoding
- RTSP streaming
- HTTP control interface

9.14 imageaibrokerpy

`imageaibroker.py` application demonstrates how to use AI/ML detection algorithms together with Python API of MRTech IFF SDK. It is located in `samples/08_imageai_py` directory of IFF SDK package. Application comes with example configuration file (`imageaibroker.json`) providing the following functionality:

- acquisition from XIMEA camera
- 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
- image export to the user code

Additionally example code uses ImageAI Python library to detect objects and OpenCV library to render images on the screen, both of which should be installed in the system.

A Changelog

A.1 Version 2.0.1

- Added `imageaibrokerpy` sample Python application.
- Fixed occasional crash on Linux when creating multiple `cuda_processor` elements.

A.2 Version 2.0

- Added `frame_importer` component with corresponding C API functions and `imagefilter` sample application demonstrating their usage.
- Introduced asynchronous mode to `frame_exporter` component.
- Added IFF SDK library C++ wrapper and corresponding sample applications.
- Added IFF SDK Python bindings and corresponding sample applications.
- Introduced runtime control of `color` parameters in `cuda_processor` component.
- Enhanced auto-gain algorithm in `awb_aec` component.
- Enhanced compatibility of `genicam` component with further machine vision camera vendors (Baumer and The Imaging Source cameras were additionally tested).
- Renamed `P010_10LE` pixel format to `P010`.
- Renamed `auto_wb` parameter to `auto_white_balance` in `xicamera` component.
- Introduced control of message title to `iff_log()` function.
- Command execution operation now returns a JSON result.
- All `iff_callback_t` callbacks are now called from a separate thread sequentially.
- All `iff_callback_t` and `iff_image_handler_t` callbacks are now guaranteed to never be called after corresponding `iff_release_chain()` function call returns.
- Added `iff_error_code` enumeration to C header file.
- Documented special read-only parameters of `genicam`, `v4l2cam` and `xicamera` components.

A.2.1 Migration guide

Some incompatible changes were introduced in this major update. To adapt your code from 1.x IFF SDK version consult the following list of changes:

- `iff_log` now takes `const char* title` as a new second parameter (pass `"IFFUSEROUT"` to keep the old behaviour);
- `iff_result_handler_t` type was removed (`iff_callback_t` is now used instead);
- `iff_frame_export_function_t` type was renamed to `iff_image_handler_t`;
- `iff_callback_t` (used in `iff_get_params` and `iff_set_callback` functions) and `iff_error_handler_t` (used in `iff_create_chain` function) callbacks now take `void* private_data` as a new last parameter (can be ignored to keep the old behaviour);

- `iff_create_chain`, `iff_get_params` and `iff_set_callback` functions now take `void* private_data` as a new last parameter (pass `nullptr` to keep the old behaviour);
- `iff_execute` function now takes `iff_callback_t result_handler, void* private_data` as new last parameters (pass `[](const char*, void*){}, nullptr` to keep the old behaviour);
- some `iff_image_metadata` (used in `iff_image_handler_t` callback for `iff_set_export_callback` function) fields have been renamed or re-ordered:
  - `ts` is now `src_ts`;
  - `ntp_time` is now `ntp_ts`;
  - `padding` moved from the first position to the last;
  - `exposure` and `black_level` were swapped.

Alternatively it can also be a good time to switch to IFF SDK library C++ wrapper, which is more convenient to use from C++ code.

A.3 Version 1.9

- Added `undistort` filter to `cuda_processor` component (available only on Linux, Windows support to come later).
- Added `lensprofiler` distortion estimation tool to sample applications.
- Added `streamadapter` GStreamer integration sample.
- Introduced auto-gain functionality to `awb_aec` component.
- Replaced `wait_limit` parameter with `ctrl_latency` in `awb_aec` component.
- Added `linear_gain` parameter to `v4l2cam` component.
- Simplified usage of `cuda_processor` export adapters (`export_to_devmem` and `export_to_hostmem` were merged into `export_to_device` and `export_to_host` respectively).
- Enhanced rendering code in `imagebroker` sample application, decreasing CPU load and latency.
- Expanded `encoder` component functionality to support encoding of `Mono16` images with 10-bit H265 codec on NVIDIA Jetson platform.
- Introduced possibility to set white balance as light source temperature to all source components (except `rtsp_source`).
- Various minor bug fixes and documentation improvements.

A.4 Version 1.8.1

- Enhanced compatibility of `genicam` component with various machine vision camera vendors (Basler, LUCID and XIMEA cameras were tested).
- Improved reliability of image metadata produced by `genicam` component in case of runtime modification of exposure time or gain (e.g. by `awb_aec` component).
- Improved detection of GigE Vision camera disconnection by `genicam` component.

A.5 Version 1.8

- Added exr_writer, fps_limiter, gamma, highlight_recovery, packer and resizer components.
- Added exposure_indicator, huesatmap, denoise and raw_denoise filters to cuda_processor component.
- Added spectraprofiler color profiling tool to sample applications.
- Introduced color profile support to cuda_processor and xiprocessor components.
- Introduced multi-render functionality to imagebroker sample application and improved its overall performance.
- Expanded NVIDIA Jetson Linux (L4T) support up to version 36.
- Fixed video artifacts in encoder output on NVIDIA Jetson Orin platform.
- Improved compatibility of genicam component with various GenICam cameras.
- Added black_level and max_buffers_queue_size parameters to genicam component.
- Added possibility to change ev_correction parameter of awb_aec component at runtime.
- Added new parameters (tags) to dng_writer component.
- Added support for 16-bit RGB formats to xicamera and xiprocessor components.
- Enhanced filename template parameter in raw_frame_player, frames_writer and dng_writer components.
- Renamed cam_ts metadata field to src_ts in metadata_saver and raw_frame_player components.
- Fixed output directory path calculation in frames_writer and dng_writer components in case of empty base_directory parameter and non-empty subdirectory parameter in on command.
- Fixed auto-start functionality in some sinks.
- Fixed hang on exit in case some sinks are still on.
- Corrected default value of direct_io parameter in files_writer component documentation.
- Various minor bug fixes and documentation improvements.

A.6 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.7 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.8 Version 1.5

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

A.9 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.10 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.11 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.12 Version 1.1

No functional changes, only documentation update.

A.13 Version 1.0

Initial release.

B License notices

AVIR

IFF SDK uses AVIR library under the MIT License.

> Copyright (c) 2015-2021 Aleksey Vaneev
>
> 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.

Boost

IFF SDK uses Boost C++ libraries under the Boost Software License, Version 1.0.

{fmt}

IFF SDK uses {fmt} library under the MIT License.

GenICam®

GenICam edition of IFF SDK uses GenICam libraries under the 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.

The GNU C Library

Linux version of IFF SDK uses glibc library under the GNU Lesser General Public License.

The GNU Compiler Collection

Linux version of IFF SDK uses libgcc and libstdc++ libraries under the GNU General Public License plus the GCC Runtime Library Exception.

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.

Microsoft Visual C++ Runtime

Windows version of IFF SDK uses Microsoft Visual C++ Runtime libraries under the Microsoft Software License Terms.

NVIDIA® CUDA® Toolkit

CUDA edition of IFF SDK uses CUDA Toolkit under the EULA.

OpenEXR

IFF SDK uses OpenEXR and Imath libraries under the BSD-3-Clause License.

> Copyright (c) Contributors to the OpenEXR Project. All rights reserved.
>
> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
> 2. 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.
> 3. Neither the name of the copyright holder 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

pybind11

IFF SDK uses pybind11 library under the BSD-3-Clause License.

xiAPI

XIMEA edition of IFF SDK uses XIMEA Application Programming Interface (xiAPI) under the License For Customer Use of XIMEA Software.