

---

# DepthAI API Docs

*Release*

***2.1.0.0.dev+a60bdfb9c189e17d2356728675fd91be9a5c8c7e***

**Luxonis**

**Mar 29, 2021**



## CONTENT:

<b>1</b>	<b>What is Gen2?</b>	<b>3</b>
<b>2</b>	<b>Basic glossary</b>	<b>5</b>
<b>3</b>	<b>Getting started</b>	<b>7</b>
3.1	Installation . . . . .	7
3.2	Hello World . . . . .	12
3.3	Multiple DepthAI per Host . . . . .	16
3.4	Local OpenVINO Model Conversion . . . . .	18
3.5	01 - RGB Preview . . . . .	26
3.6	02 - Mono Preview . . . . .	27
3.7	03 - Depth Preview . . . . .	28
3.8	04 - RGB Encoding . . . . .	31
3.9	05 - RGB & Mono Encoding . . . . .	32
3.10	06 - RGB Full Resolution Saver . . . . .	34
3.11	07 - Mono Full Resolution Saver . . . . .	36
3.12	08 - RGB & MobilenetSSD . . . . .	37
3.13	09 - Mono & MobilenetSSD . . . . .	40
3.14	10 - Mono & MobilenetSSD & Encoding . . . . .	42
3.15	11 - RGB & Encoding & Mono & MobilenetSSD . . . . .	46
3.16	12 - RGB Encoding & Mono with MobilenetSSD & Depth . . . . .	49
3.17	13 - Encoding Max Limit . . . . .	53
3.18	14 - Color Camera Control . . . . .	55
3.19	15 - 4K RGB MobileNetSSD . . . . .	59
3.20	16 - Device Queue Event . . . . .	62
3.21	17 - Video & MobilenetSSD . . . . .	63
3.22	18 - RGB Encoding with MobilenetSSD . . . . .	66
3.23	21 - RGB & MobilenetSSD decoding on device . . . . .	68
3.24	22.1 - RGB & TinyYoloV3 decoding on device . . . . .	69
3.25	22.2 - RGB & TinyYoloV4 decoding on device . . . . .	72
3.26	23 - Auto Exposure on ROI . . . . .	75
3.27	24 - OpenCV support . . . . .	80
3.28	25 - System information . . . . .	81
3.29	26.1 - RGB & MobilenetSSD with spatial data . . . . .	82
3.30	26.2 - MONO & MobilenetSSD with spatial data . . . . .	86
3.31	26.1 - RGB & TinyYolo with spatial data . . . . .	90
3.32	27 - Spatial location calculator . . . . .	95
3.33	28 - Camera video high resolution . . . . .	98
3.34	Python API Reference . . . . .	99
3.35	C++ API Reference . . . . .	207

<b>Python Module Index</b>	<b>261</b>
<b>Index</b>	<b>263</b>

On this page you can find the details regarding the Gen2 DepthAI API that will allow you to interact with the DepthAI device. We support both *Python API* and *C++ API*



## **WHAT IS GEN2?**

Gen2 is a step forward in DepthAI integration, allowing users to define their own flow of data using pipelines, nodes and connections. Gen2 was created based on user's feedback from Gen1 and from raising capabilities of both DepthAI and supporting software like OpenVINO.





## BASIC GLOSSARY

- **Host side** is the device, like PC or RPi, to which the DepthAI is connected to. If something is happening on the host side, it means that this device is involved in it, not DepthAI itself
- **Device side** is the DepthAI itself. If something is happening on the device side, it means that the DepthAI is responsible for it
- **Pipeline** is a complete workflow on the device side, consisting of nodes and connections between them - these cannot exist outside of pipeline.
- **Node** is a single functionality of the DepthAI. It have either inputs or outputs or both, together with properties to be defined (like resolution on the camera node or blob path in neural network node)
- **Connection** is a link between one node's output and another one's input. In order to define the pipeline dataflow, the connections define where to send data in order to achieve an expected result
- **XLink** is a middleware that is capable to exchange data between device and host. XLinkIn node allows to send the data from host to device, XLinkOut does the opposite.



## GETTING STARTED

To help you get started with Gen2 API, we have prepared multiple examples of it's usage, with more yet to come, together with some insightful tutorials.

Before running the example, install the DepthAI Python library using the command below

```
python3 -m pip install --extra-index-url https://artifacts.luxonis.com/artifactory/  
↳luxonis-python-snapshot-local/ depthai==2.1.0.0.  
↳dev+a60bdfb9c189e17d2356728675fd91be9a5c8c7e
```

Now, pick a tutorial or code sample and start utilizing Gen2 capabilities

### 3.1 Installation

Please *install the necessary dependencies* for your platform by referring to the table below. Once installed you can *install the DepthAI library*.

We are constantly striving to improve how we release our software to keep up with countless platforms and the numerous ways to package it. If you do not see a particular platform or package format listed below please reach out to us on [Discord](#) or on [Github](#).

#### 3.1.1 Supported Platforms

We keep up-to-date, pre-compiled, libraries for the following platforms. Note that a new change is that for Ubuntu now also work unchanged for the Jetson/Xavier series:

Platform	Instructions	Tutorial	Support
Windows 10	<a href="#">Platform dependencies</a>	<a href="#">Video tutorial</a>	<a href="#">Discord</a>
macOS	<a href="#">Platform dependencies</a>	<a href="#">Video tutorial</a>	<a href="#">Discord</a>
Ubuntu & Jetson/Xavier	<a href="#">Platform dependencies</a>	<a href="#">Video tutorial</a>	<a href="#">Discord</a>
Raspberry Pi	<a href="#">Platform dependencies</a>	<a href="#">Video tutorial</a>	<a href="#">Discord</a>

And the following platforms are also supported by a combination of the community and Luxonis.

Platform	Instructions	Support
Fedora		<a href="#">Discord</a>
Robot Operating System		<a href="#">Discord</a>
Windows 7	<a href="#">WinUSB driver</a>	<a href="#">Discord</a>
Docker	<a href="#">Pull and run official images</a>	<a href="#">Discord</a>

## macOS

```
bash -c "$(curl -fL http://docs.luxonis.com/_static/install_dependencies.sh)"
```

Close and re-open the terminal window after this command.

The script also works on M1 Macs, Homebrew being installed under Rosetta 2, as some Python packages are still missing native M1 support. In case you already have Homebrew installed natively and things don't work, see [here](#) for some additional troubleshooting steps.

Note that if the video streaming window does not appear consider running the following:

```
python3 -m pip install opencv-python --force-reinstall --no-cache-dir
```

See the [Video preview window fails to appear on macOS](#) thread on our forum for more information.

## Raspberry Pi OS

```
sudo curl -fL http://docs.luxonis.com/_static/install_dependencies.sh | bash
```

## Ubuntu

These Ubuntu instructions also work for the **Jetson** and **Xavier** series.

```
sudo wget -qO- http://docs.luxonis.com/_static/install_dependencies.sh | bash
```

Note! If opencv fails with illegal instruction after installing from PyPi, add:

```
echo "export OPENBLAS_CORETYPE=ARMV8" >> ~/.bashrc
source ~/.bashrc
```

## openSUSE

For openSUSE, available in [this official article](#) how to install the OAK device on the openSUSE platform.

## Windows

We recommend using the Chocolatey package manager to install DepthAI's dependencies on Windows. Chocolatey is very similar to Homebrew for macOS. Alternatively, it is also possible to install DepthAI and its dependencies manually, although it can be more time consuming and error prone.

To [install Chocolatey](#) and use it to install DepthAI's dependencies do the following:

- Right click on *Start*
- Choose *Windows PowerShell (Admin)* and run the following:

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.
↪ServicePointManager]::SecurityProtocol = [System.Net.
↪ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.
↪WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

- Close the PowerShell and then re-open another PowerShell (Admin) by repeating the first two steps.

- Install Python and PyCharm

```
choco install cmake git python pycharm-community -y
```

## Windows 7

Although we do not officially support Windows 7, members of the community [have had success](#) manually installing WinUSB using [Zadig](#). After connecting your DepthAI device look for a device with USB ID: 03E7 2485 and install the WinUSB driver by selecting *WinUSB(v6.1.7600.16385)* and then *Install WCID Driver*.

## Docker

We maintain a Docker image containing DepthAI, it's dependencies and helpful tools in the [luxonis/depthai-library](#) repository on Docker Hub. It builds upon the [luxonis/depthai-base](#) image.

Run the `01_rgb_preview.py` example inside a Docker container on a Linux host (with the X11 windowing system):

```
docker pull luxonis/depthai-library
docker run --rm \
  --privileged \
  -v /dev/bus/usb:/dev/bus/usb \
  --device-cgroup-rule='c 189:* rmw' \
  -e DISPLAY=$DISPLAY \
  -v /tmp/.X11-unix:/tmp/.X11-unix \
  luxonis/depthai-library:latest \
  python3 /depthai-python/examples/01_rgb_preview.py
```

To allow the container to update X11 you may need to run `xhost local:root` on the host.

### 3.1.2 Install from PyPI

Our packages are distributed [via PyPi](#), to install it in your environment use

```
python3 -m pip install depthai
```

For other installation options, see [other installation options](#).

### 3.1.3 Test installation

We have [a set of examples](#) that should help you verify if your setup was correct.

First, clone the [depthai-python](#) repository and change directory into this repo:

```
git clone https://github.com/luxonis/depthai-python.git
cd depthai-python
```

Next install the requirements for this repository. Note that we recommend installing the dependencies in a virtual environment, so that they don't interfere with other Python tools/environments on your system.

- For development machines like Mac/Windows/Ubuntu/etc., we recommend the [PyCharm](#) IDE, as it automatically makes/manages virtual environments for you, along with a bunch of other benefits. Alternatively, `conda`, `pipenv`, or `virtualenv` could be used directly (and/or with your preferred IDE).

- For installations on resource-constrained systems, such as the Raspberry Pi or other small Linux systems, we recommend `conda`, `pipenv`, or `virtualenv`. To set up a virtual environment with `virtualenv`, run `virtualenv venv && source venv/bin/activate`.

Using a virtual environment (or system-wide, if you prefer), run the following to install the requirements for this example repository:

```
cd examples
python3 install_requirements.py
```

Now, run the `01_rgb_preview.py` script from within `examples` directory to make sure everything is working:

```
python3 01_rgb_preview.py
```

If all goes well a small window video display should appear. An example is shown below:

### 3.1.4 Run Other Examples

After you have run this example, you can run other examples to learn about DepthAI possibilities. You can also proceed to:

- Our tutorials, starting with a Hello World tutorial explaining the API usage step by step ([here](#))
- Our experiments, containing implementations of various user use cases on DepthAI ([here](#))

You can also proceed below to learn how to convert your own neural network to run on DepthAI.

And we also have online model training below, which shows you how to train and convert models for DepthAI:

- Online ML Training and model Conversion: [HERE](#)

### 3.1.5 Other installation methods

To get the latest and yet unreleased features from our source code, you can go ahead and compile `depthai` package manually.

#### Dependencies to build from source

- CMake > 3.2.0
- Generation tool (Ninja, make, ...)
- C/C++ compiler
- libusb1 development package

#### Ubuntu, Raspberry Pi OS, ... (Debian based systems)

On Debian based systems (Raspberry Pi OS, Ubuntu, ...) these can be acquired by running:

```
sudo apt-get -y install cmake libusb-1.0-0-dev build-essential
```

## macOS (Mac OS X)

Assuming a stock Mac OS X install, `depthai-python` library needs following dependencies

- Homebrew (If it's not installed already)

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/  
↪master/install.sh)"
```

- Python, libusb, CMake, wget

```
brew install coreutils python3 cmake libusb wget
```

And now you're ready to clone the `depthai-python` from Github and build it for Mac OS X.

### Install using GitHub commit

Pip allows users to install the packages from specific commits, even if they are not yet released on PyPi.

To do so, use the command below - and be sure to replace the `<commit_sha>` with the correct commit hash [from here](#)

```
python3 -m pip install git+https://github.com/luxonis/depthai-python.git@<commit_sha>
```

### Using/Testing a Specific Branch/PR

From time to time, it may be of interest to use a specific branch. This may occur, for example, because we have listened to your feature request and implemented a quick implementation in a branch. Or it could be to get early access to a feature that is soaking in our `develop` for stability purposes before being merged into `main` (`develop` is the branch we use to soak new features before merging them into `main`):

So when working in the `depthai-python` repository, using a branch can be accomplished with the following commands.

Prior to running the following, you can either clone the repository independently (for not over-writing any of your local changes) or simply do a `git pull` first.

```
git checkout <branch>  
git submodule update --init --recursive  
python3 setup.py develop
```

### Install from source

If desired, you can also install the package from the source code itself - it will allow you to make the changes to the API and see them live in action.

To do so, first download the repository and then add the package to your python interpreter in development mode

```
git clone https://github.com/luxonis/depthai-python.git  
cd depthai-python  
git submodule update --init --recursive  
python3 setup.py develop # you may need to add sudo if using system interpreter  
↪instead of virtual environment
```

If you want to use other branch (e.g. `develop`) than default (`main`), you can do so by typing

```
git checkout develop # replace the "develop" with a desired branch name
git submodule update --recursive
python3 setup.py develop
```

Or, if you want to checkout a specific commit, type

```
git checkout <commit_sha>
git submodule update --recursive
python3 setup.py develop
```

## 3.2 Hello World

Learn how to use the DepthAI Python API to display a color video stream.

### 3.2.1 Demo

### 3.2.2 Dependencies

Let's get your development environment setup first. This tutorial uses:

- Python 3.6 (Ubuntu) or Python 3.7 (Raspbian).
- The DepthAI Python API
- The `cv2` and `numpy` Python modules.

### 3.2.3 Code Overview

The `depthai` Python module provides access to your board's 4K 60 Hz color camera. We'll display a video stream from this camera to your desktop. You can find the [complete source code for this tutorial on GitHub](#).

### 3.2.4 File Setup

Setup the following file structure on your computer:

```
cd ~
mkdir -p depthai-tutorials-practice/1-hello-world
touch depthai-tutorials-practice/1-hello-world/hello_world.py
cd depthai-tutorials-practice/1-hello-world
```

What's with the `-practice` suffix in parent directory name? Our tutorials are available on GitHub via the [depthai-tutorials](#) repository. We're appending `-practice` so you can distinguish between your work and our finished tutorials (should you choose to download those).



### 3.2.5 Install pip dependencies

To display the DepthAI color video stream we need to import a small number of packages. Download and install the requirements for this tutorial:

```
python3 -m pip install numpy opencv-python depthai --user
```

### 3.2.6 Test your environment

Let's verify we're able to load all of our dependencies. Open the `hello_world.py` file you *created earlier* in your code editor. Copy and paste the following into `hello_world.py`:

```
import numpy as np # numpy - manipulate the packet data returned by depthai
import cv2 # opencv - display the video stream
import depthai # access the camera and its data packets
```

Try running the script and ensure it executes without error:

```
python3 hello_world.py
```

If you see the following error:

```
ModuleNotFoundError: No module named 'depthai'
```

...follow [these steps](#) in our troubleshooting section.

### 3.2.7 Define a pipeline

Any action from DepthAI, whether it's a neural inference or color camera output, require a **pipeline** to be defined, including nodes and connections corresponding to our needs.

In this case, we want to see the frames from **color camera**, as well as a simple **neural network** to be ran on top of them.

Let's start off with an empty Pipeline object

```
pipeline = depthai.Pipeline()
```

Now, first node we will add is a ColorCamera. We will use the preview output, resized to 300x300 to fit the **mobilenet-ssd input size** (which we will define later)

```
cam_rgb = pipeline.createColorCamera()
cam_rgb.setPreviewSize(300, 300)
cam_rgb.setInterleaved(False)
```

Up next, let's define a NeuralNetwork node with **mobilenet-ssd network**. The blob file for this example can be found [here](#)

```
detection_nn = pipeline.createNeuralNetwork()
detection_nn.setBlobPath("/path/to/mobilenet-ssd.blob")
```

And now, let's connect a color camera preview output to neural network input

```
cam_rgb.preview.link(detection_nn.input)
```

Finally, we want to receive both color camera frames and neural network inference results - as these are produced on the device, they need to be transported to our machine (host). The communication between device and host is handled by XLink, and in our case, since we want to receive data from device to host, we will use XLinkOut node

```
xout_rgb = pipeline.createXLinkOut()
xout_rgb.setStreamName("rgb")
cam_rgb.preview.link(xout_rgb.input)

xout_nn = pipeline.createXLinkOut()
xout_nn.setStreamName("nn")
detection_nn.out.link(xout_nn.input)
```

### 3.2.8 Initialize the DepthAI Device

Having the pipeline defined, we can now initialize a device and start it

```
device = depthai.Device(pipeline)
device.startPipeline()
```

---

**Note:** By default, the DepthAI is accessed as a USB3 device. This comes with [several limitations](#).

If you'd like to communicate via USB2, being free from these but having a limited bandwidth, initialize the DepthAI with the following code

```
device = depthai.Device(pipeline, True)
```

---

From this point on, the pipeline will be running on the device, producing results we requested. Let's grab them

### 3.2.9 Adding helpers

As XLinkOut nodes has been defined in the pipeline, we'll define now a host side output queues to access the produced results

```
q_rgb = device.getOutputQueue("rgb")
q_nn = device.getOutputQueue("nn")
```

These will fill up with results, so next thing to do is consume the results. We will need two placeholders - one for rgb frame and one for nn results

```
frame = None
bboxes = []
```

Also, due to neural network implementation details, bounding box coordinates in inference results are represented as floats from <0..1> range - so relative to frame width/height (e.g. if image has 200px width and nn returned x\_min coordinate equal to 0.2, this means the actual (normalised) x\_min coordinate is 40px).

That's why we need to define a helper function, `frame_norm`, that will convert these <0..1> values into actual pixel positions

```
def frame_norm(frame, bbox):
    return (np.array(bbox) * np.array([*frame.shape[:2], *frame.shape[:2]])[::-1]).
    ↪ astype(int)
```

### 3.2.10 Consuming the results

Having everything prepared, we are ready to start out main program loop

```
while True:
    # ...
```

Now, inside this loop, first thing to do is fetching latest results from both nn node and color camera

```
in_rgb = q_rgb.tryGet()
in_nn = q_nn.tryGet()
```

The `tryGet` method returns either the latest result or `None` if the queue is empty.

Results, both from rgb camera or neural network, will be delivered as 1D arrays, so both of them will require transformations to be useful for display (we have already defined one of the transformations needed - the `frame_norm` function)

First up, if we receive a frame from rgb camera, we need to convert it from 1D array into HWC form (HWC stands for Height Width Channels, so 3D array, with first dimension being width, second height, and third the color channel)

```
if in_rgb is not None:
    shape = (3, in_rgb.getHeight(), in_rgb.getWidth())
    frame = in_rgb.getData().reshape(shape).transpose(1, 2, 0).astype(np.uint8)
    frame = np.ascontiguousarray(frame)
```

Second, the neural network results will also need transformations. These are also returned as a 1D array, but this time the array has a fixed size (constant, no matter how many results the neural network has actually produced). Actual results in array are followed with `-1` and then filled to meet the fixed size with `0`. One results has 7 fields, each being respectively `image_id`, `label`, `confidence`, `x_min`, `y_min`, `x_max`, `y_max`. We will want only the last four values (being the bounding box), but we'll also filter out the ones which confidence is below a certain threshold - it can be anywhere between `<0.1`, and for this example we will use `0.8` threshold

```
if in_nn is not None:
    bboxes = np.array(in_nn.getFirstLayerFp16())
    bboxes = bboxes[:np.where(bboxes == -1)[0][0]]
    bboxes = bboxes.reshape((bboxes.size // 7, 7))
    bboxes = bboxes[bboxes[:, 2] > 0.8][:, 3:7]
```

To better understand this flow, let's take an example. Let's assume the `np.array(in_nn.getFirstLayerFp16())` returns the following array

```
[0, 15, 0.99023438, 0.45556641, 0.34399414 0.88037109, 0.9921875, 0, 15, 0.98828125,
↪0.03076172, 0.23388672, 0.60205078, 1.0078125, -1, 0, 0, 0, ...]
```

First operation, `bboxes[:np.where(bboxes == -1)[0][0]]`, removes the trailing zeros from the array, so now the bbox array will look like this

```
[0, 15, 0.99023438, 0.45556641, 0.34399414 0.88037109, 0.9921875, 0, 15, 0.98828125,
↪0.03076172, 0.23388672, 0.60205078, 1.0078125]
```

Second one - `bboxes.reshape((bboxes.size // 7, 7))`, reshapes the 1D array into 2D array - where each row is a separate result

```
[
  [0, 15, 0.99023438, 0.45556641, 0.34399414 0.88037109, 0.9921875],
  [0, 15, 0.98828125, 0.03076172, 0.23388672, 0.60205078, 1.0078125]
]
```

Last one - `bboxes = bboxes[bboxes[:, 2] > 0.8][:, 3:7]` - will filter the results based on the confidence column (3rd one, with index 2) to be above a defined threshold (0.8) - and from these results, it will only take the last 4 columns being the bounding boxes. Since both our results have a very high confidence (0.99023438 and 0.98828125 respectively), they won't be filtered, and the final array will look like this

```
[
  [0.45556641, 0.34399414, 0.88037109, 0.9921875],
  [0.03076172, 0.23388672, 0.60205078, 1.0078125]
]
```

### 3.2.11 Display the results

Up to this point, we have all our results consumed from the DepthAI device, and only thing left is to actually display them.

```
if frame is not None:
    for raw_bbox in bboxes:
        bbox = frame_norm(frame, raw_bbox)
        cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
    cv2.imshow("preview", frame)
```

You can see here the usage of `frame_norm` we defined earlier for bounding box coordinates normalization. By using `cv2.rectangle` we draw a rectangle on the rgb frame as an indicator where the face position is, and then we display the frame using `cv2.imshow`

Finally, we add a way to terminate our program (as it's running inside an infinite loop). We will use `cv2.waitKey` method, that waits for a key to be pressed by user - in our case, we want to break out of the loop when user presses `q` key

```
if cv2.waitKey(1) == ord('q'):
    break
```

### 3.2.12 Running the example

Putting it all together, only thing left to do is to run the file we've prepared in this tutorial and see the results

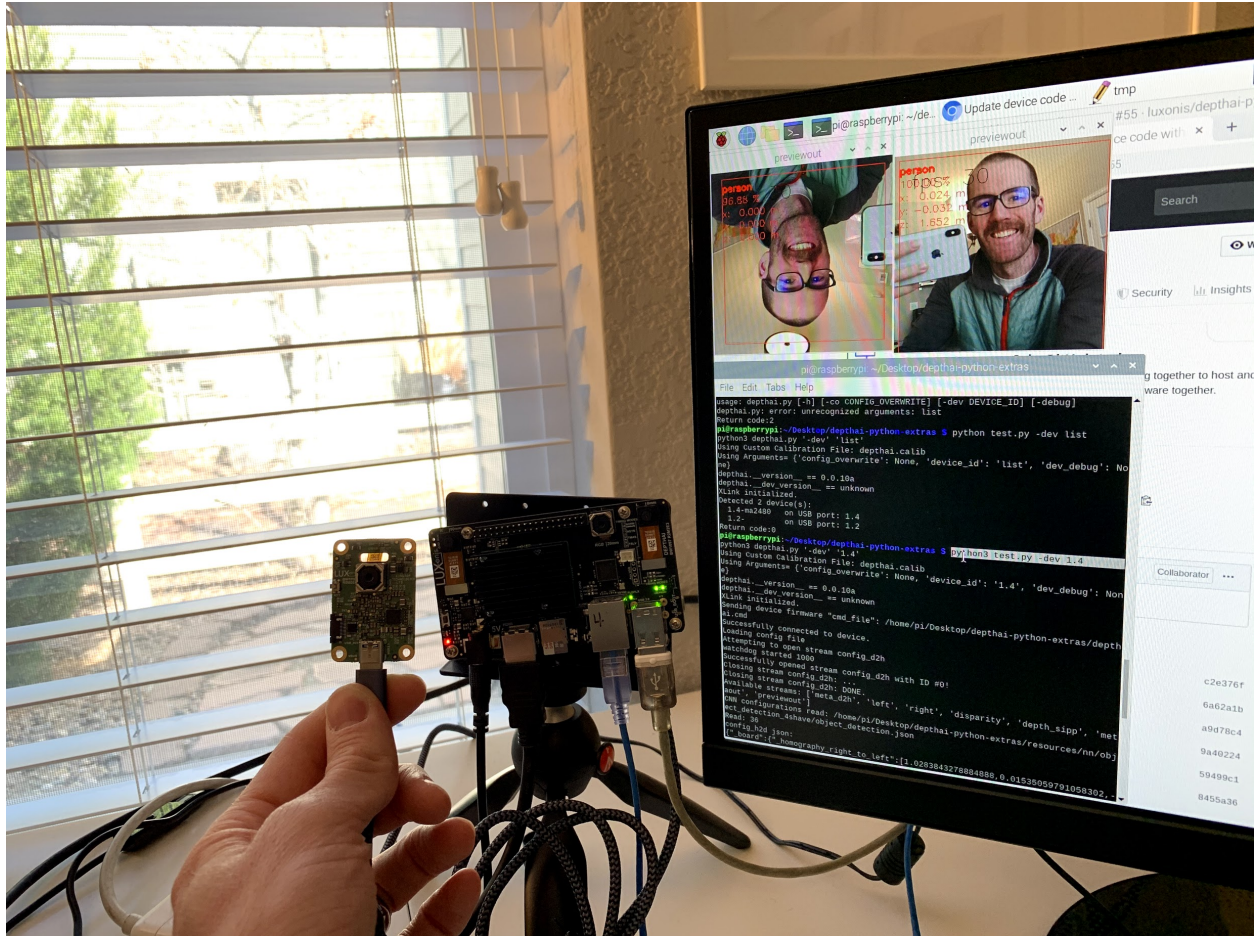
```
python3 hello_world.py
```

You're on your way! You can find the [complete code for this tutorial on GitHub](#).

We're always happy to help with code or other questions you might have.

## 3.3 Multiple DepthAI per Host

Learn how to discover DepthAI devices connected to your system, and use them individually.



Shown on the left is Luxonis **uAI (BW1093)** which is actually plugged into a **Raspberry Pi Compute Module Edition (BW1097)**.

So in this case, everything is running on the (single) Raspberry Pi 3B+ which is in the back of the BW1097.

### 3.3.1 Dependencies

You have already set up the Python API on your system (if you have a Raspberry Pi Compute Module it came pre-setup). See [here](#) if you have not yet installed the DepthAI Python API on your system.

### 3.3.2 Discover DepthAI-USB Port Mapping

The DepthAI multi-device support is currently done by selecting the device `mx_id` (serial number) of a connected DepthAI device.

If you'd like to associate a given DepthAI device with specific code (e.g. neural model) to be run on it, it is recommended to plug in one device at a time, and then use the following code to determine which device is on which port:

```
import depthai
for device in depthai.Device.getAllAvailableDevices():
    print(f"{device.getMxId()} {device.state}")
```

Example results for 2x DepthAI on a system:

## 3.3. Multiple DepthAI per Host

```
14442C10D13EABCE00 XLinkDeviceState.X_LINK_UNBOOTED
14442C1071659ACD00 XLinkDeviceState.X_LINK_UNBOOTED
```

### 3.3.3 Selecting a Specific DepthAI device to be used.

From the Detected devices(s) above, use the following code to select the device you would like to use with your pipeline. For example, if the first device is desirable from above use the following code:

```
found, device_info = depthai.Device.getDeviceByMxId("14442C10D13EABCE00")

if not found:
    raise RuntimeError("Device not found!")
```

You can then use the *device\_info* to specify on which device you want to run your pipeline:

```
with depthai.Device(pipeline, device_info) as device:
```

And you can use this code as a basis for your own use cases, such that you can run differing neural models on different DepthAI/uAI models.

Now use as many DepthAI devices as you need!

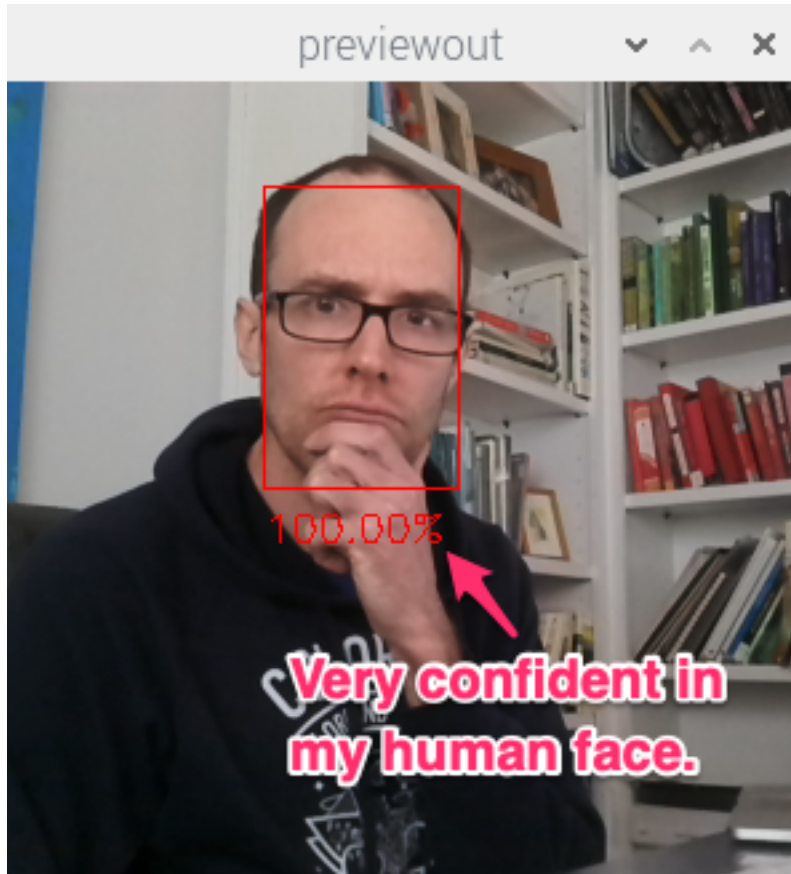
And since DepthAI does all the heavy lifting, you can usually use quite a few of them with very little burden to the host.

We're always happy to help with code or other questions you might have.

## 3.4 Local OpenVINO Model Conversion

In this tutorial, you'll learn how to convert OpenVINO IR models into the format required to run on DepthAI, even on a low-powered Raspberry Pi. I'll introduce you to the OpenVINO toolset, the Open Model Zoo (where we'll download the [face-detection-retail-0004](#) model), and show you how to generate the files needed to run model inference on your DepthAI board.





Haven't heard of OpenVINO or the Open Model Zoo? I'll start with a quick introduction of why we need these tools.

### 3.4.1 What is OpenVINO?

Under-the-hood, DepthAI uses the Intel technology to perform high-speed model inference. However, you can't just dump your neural net into the chip and get high-performance for free. That's where [OpenVINO](#) comes in. OpenVINO is a free toolkit that converts a deep learning model into a format that runs on Intel Hardware. Once the model is converted, it's common to see Frames Per Second (FPS) improve by 25x or more. Are a couple of small steps worth a 25x FPS increase? Often, the answer is yes!

### 3.4.2 What is the Open Model Zoo?

The [Open Model Zoo](#) is a library of freely-available pre-trained models. The Zoo also contains scripts for downloading those models into a compile-ready format to run on DepthAI.

DepthAI is able to run many of the object detection models in the Zoo.

### 3.4.3 Install OpenVINO

**Warning:** If you have OpenVINO installed or want to follow [official installation](#), skip this step.

Please note that the following install instructions are for **Ubuntu 18.04** OS, if you intend to use other OS, follow the official OpenVINO installation

DepthAI requires OpenVINO version 2020.1. Let's get a package for our OS and meeting this version with the following command:

```
apt-get update
apt-get install -y software-properties-common
add-apt-repository -y ppa:deadsnakes/ppa
apt-get update
apt-get install -y wget pciutils python3.8 libpng-dev libcairo2-dev libpango1.0-dev \
↳ libglib2.0-dev libgtk2.0-dev libswscale-dev libavcodec-dev libavformat-dev
cd
mkdir openvino_install && cd openvino_install
wget http://registrationcenter-download.intel.com/akdlm/irc_nas/16345/l_openvino_
↳ toolkit_p_2020.1.023.tgz
tar --strip-components=1 -zxvf l_openvino_toolkit_p_2020.1.023.tgz
./install_openvino_dependencies.sh
./install.sh # when finished, you can go ahead and do "rm -r ~/openvino_install"
```

Now, first screen we'll see is EULA, just hit Enter, scroll through and type accept.

Next one is agreement to Intel Software Improvement Program, it's not relevant so you can choose whether consent (1) or not (2)

Next, you may see the Missing Prerequisites screen showing that Intel® Graphics Compute Runtime for OpenCL™ Driver is missing - you can go ahead and ignore this warning.

Finally, we'll see the install summary - please verify that it has a correct location pointed out - /opt/intel. If all looks good, go ahead and proceed (1). If the missing prerequisites screen appears again, feel free to skip it.

Let's verify that a correct version is installed on your host. Check your version by running the following from a terminal session:

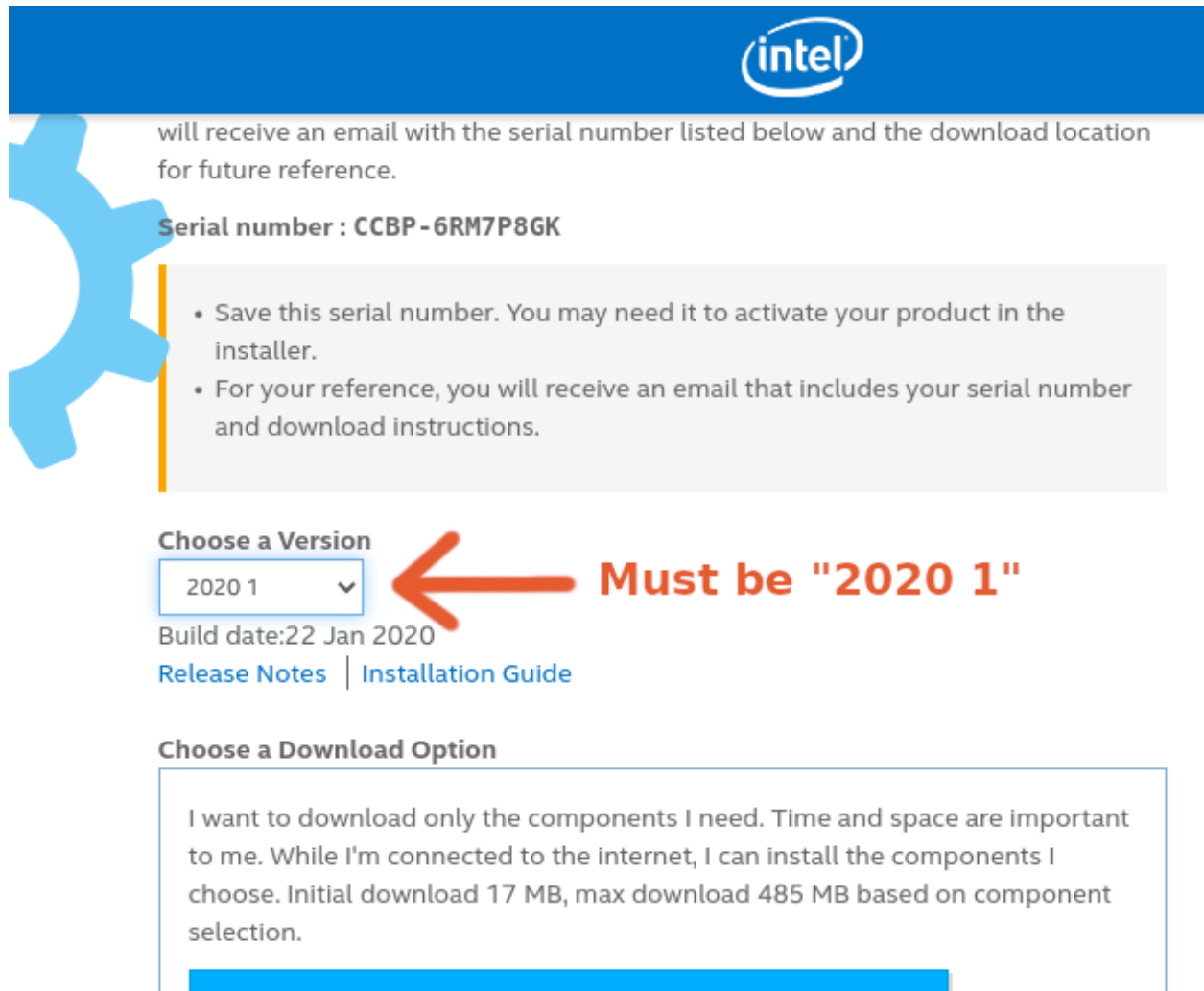
```
cat /opt/intel/openvino/inference_engine/version.txt
```

You should see output similar to:

```
Thu Jan 23 19:14:14 MSK 2020
d349c3ba4a2508be72f413fa4dee92cc0e4bc0e1
releases_2020_1_InferenceEngine_37988
```

Verify that you see releases\_2020\_1 in your output. If you do, move on. If you are on a different version, goto the [OpenVINO site](#) and download the 2020.1 version for your OS:





will receive an email with the serial number listed below and the download location for future reference.

**Serial number : CCBP-6RM7P8GK**

- Save this serial number. You may need it to activate your product in the installer.
- For your reference, you will receive an email that includes your serial number and download instructions.

**Choose a Version**

2020 1 ▼ **← Must be "2020 1"**

Build date: 22 Jan 2020

[Release Notes](#) | [Installation Guide](#)

**Choose a Download Option**

I want to download only the components I need. Time and space are important to me. While I'm connected to the internet, I can install the components I choose. Initial download 17 MB, max download 485 MB based on component selection.

### 3.4.4 Check if the Model Downloader is installed

When installing OpenVINO, you can choose to perform a smaller install to save disk space. This custom install may not include the model downloader script. Let's check if the downloader was installed. In a terminal session, type the following:

```
find /opt/intel/ -iname downloader.py
```

**Move on if you see the output below:**

```
/opt/intel/openvino_2020.1.023/deployment_tools/open_model_zoo/tools/downloader/  
↪downloader.py
```

**Didn't see any output?** Don't fret if `downloader.py` isn't found. We'll install this below.

## Install Open Model Zoo Downloader

If the downloader tools weren't found, we'll install the tools by cloning the [Open Model Zoo Repo](#) and installing the tool dependencies.

Start a terminal session and run the following commands in your terminal:

```
apt-get install -y git curl
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python3 get-pip.py
rm get-pip.py
cd ~
git clone https://github.com/opencv/open_model_zoo.git
cd open_model_zoo
git checkout tags/2020.1
cd tools/downloader
python3 -m pip install --user -r ./requirements.in
```

This clones the repo into a `~/open_model_zoo` directory, checks out the required `2020.1` version, and installs the downloader dependencies.

### 3.4.5 Create an `OPEN_MODEL_DOWNLOADER` environment variable

Typing the full path to `downloader.py` can use a lot of keystrokes. In an effort to extend your keyboard life, let's store the path to this script in an environment variable.

Run the following in your terminal:

```
export OPEN_MODEL_DOWNLOADER='INSERT PATH TO YOUR downloader.py SCRIPT'
```

Where `INSERT PATH TO YOUR downloader.py SCRIPT` can be found via:

```
find /opt/intel/ -iname downloader.py
find ~ -iname downloader.py
```

For example, if you installed `open_model_zoo` yourself:

```
export OPEN_MODEL_DOWNLOADER="$HOME/open_model_zoo/tools/downloader/downloader.py"
```

### 3.4.6 Download the face-detection-retail-0004 model

We've installed everything we need to download models from the Open Model Zoo! We'll now use the [Model Downloader](#) to download the `face-detection-retail-0004` model files. Run the following in your terminal:

```
$OPEN_MODEL_DOWNLOADER --name face-detection-retail-0004 --output_dir ~/open_model_
↳ zoo_downloads/
```

This will download the model files to `~/open_model_zoo_downloads/`. Specifically, the model files we need are located at:

```
~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16
```

You'll see two files within the directory:

```
$ ls -lh
total 1.3M
-rw-r--r-- 1 root root 1.2M Jul 28 12:40 face-detection-retail-0004.bin
-rw-r--r-- 1 root root 100K Jul 28 12:40 face-detection-retail-0004.xml
```

The model is in the OpenVINO Intermediate Representation (IR) format:

- face-detection-retail-0004.xml - Describes the network topology
- face-detection-retail-0004.bin - Contains the weights and biases binary data.

This means we are ready to compile the model for the MyriadX!

### 3.4.7 Compile the model

The MyriadX chip used on our DepthAI board does not use the IR format files directly. Instead, we need to generate face-detection-retail-0004.blob using `myriad_compile` command.

#### Locate `myriad_compile`

Let's find where `myriad_compile` is located. In your terminal, run:

```
find /opt/intel/ -iname myriad_compile
```

You should see the output similar to this

```
find /opt/intel/ -iname myriad_compile
/opt/intel/openvino_2020.1.023/deployment_tools/inference_engine/lib/intel64/myriad_
↳ compile
```

Since it's such a long path, let's store the `myriad_compile` executable in an environment variable (just like `OPEN_MODEL_DOWNLOADER`):

```
export MYRIAD_COMPILE=$(find /opt/intel/ -iname myriad_compile)
```

#### Activate OpenVINO environment

In order to use `myriad_compile` tool, we need to activate our OpenVINO environment.

First, let's find `setupvars.sh` file

```
find /opt/intel/ -name "setupvars.sh"
/opt/intel/openvino_2020.1.023/opencv/setupvars.sh
/opt/intel/openvino_2020.1.023/bin/setupvars.sh
```

We're interested in `bin/setupvars.sh` file, so let's go ahead and source it to activate the environment:

```
source /opt/intel/openvino_2020.1.023/bin/setupvars.sh
[setupvars.sh] OpenVINO environment initialized
```

If you see `[setupvars.sh] OpenVINO environment initialized` then your environment should be initialized correctly

## Run myriad\_compile

```
$MYRIAD_COMPILE -m ~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16/  
↳face-detection-retail-0004.xml -ip U8 -VPU_MYRIAD_PLATFORM VPU_MYRIAD_2480 -VPU_  
↳NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_SLICES 4
```

You should see:

```
Inference Engine:  
  API version ..... 2.1  
  Build ..... 37988  
  Description ..... API  
Done
```

Where's the blob file? It's located in the same folder as `face-detection-retail-0004.xml`:

```
ls -lh ~/open_model_zoo_downloads/intel/face-detection-retail-0004/FP16/  
total 2.6M  
-rw-r--r-- 1 root root 1.2M Jul 28 12:40 face-detection-retail-0004.bin  
-rw-r--r-- 1 root root 1.3M Jul 28 12:50 face-detection-retail-0004.blob  
-rw-r--r-- 1 root root 100K Jul 28 12:40 face-detection-retail-0004.xml
```

### 3.4.8 Run and display the model output

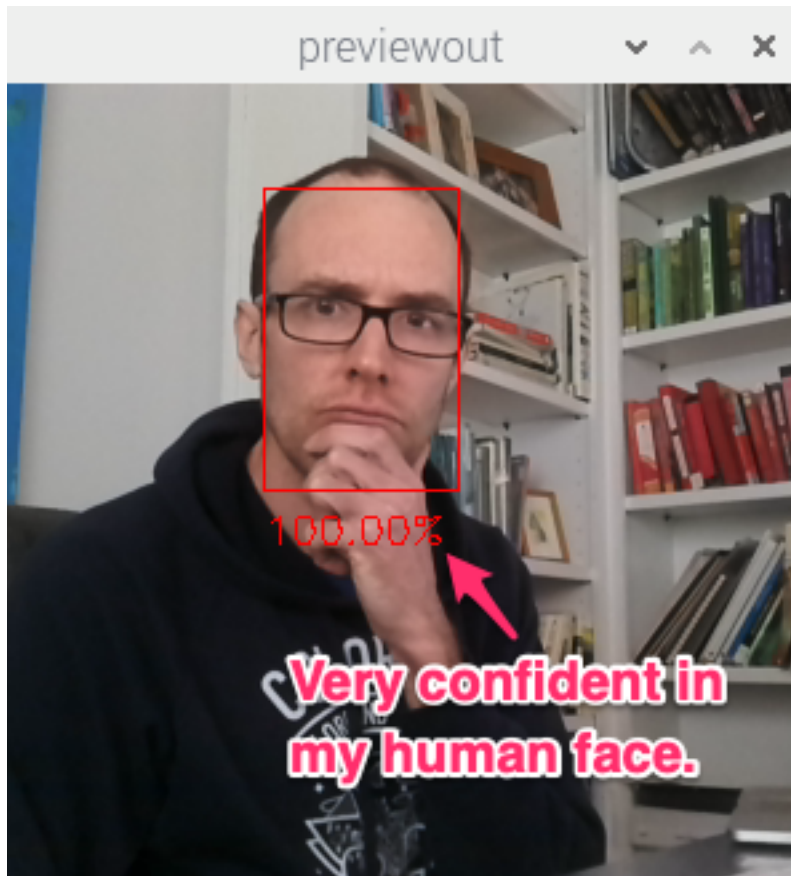
With neural network blob in place, we're ready to roll! To verify that the model is running correctly, let's modify a bit the program we've created in *Hello World* tutorial

In particular, let's change the `setBlobPath` invocation to load our model. **Remember to replace the paths to correct ones that you have!**

```
- detection_nn.setBlobPath("/path/to/mobilenet-ssd.blob")  
- detection_nn.setBlobPath("/path/to/face-detection-retail-0004.blob")
```

And that's all!

You should see output annotated output similar to:



### 3.4.9 Reviewing the flow

The flow we walked through works for other pre-trained object detection models in the Open Model Zoo:

1. Download the model:

```
$OPEN_MODEL_DOWNLOADER --name [INSERT MODEL NAME] --output_dir ~/open_  
↳model_zoo_downloads/
```

2. Create the MyriadX blob file:

```
$MYRIAD_COMPILE -m [INSERT PATH TO MODEL XML FILE] -ip U8 -VPU_MYRIAD_  
↳PLATFORM VPU_MYRIAD_2480 -VPU_NUMBER_OF_SHAVES 4 -VPU_NUMBER_OF_CMX_  
↳SLICES 4
```

3. Use this model in your script

You're on your way! You can find the [complete code for this tutorial on GitHub](#).

We're always happy to help with code or other questions you might have.

## 3.5 01 - RGB Preview

This example shows how to set up a pipeline that outputs a small preview of the RGB camera, connects over XLink to transfer these to the host real-time, and displays the RGB frames on the host with OpenCV.

### 3.5.1 Demo

### 3.5.2 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

### 3.5.3 Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  # Start defining a pipeline
7  pipeline = dai.Pipeline()
8
9  # Define a source - color camera
10 camRgb = pipeline.createColorCamera()
11 camRgb.setPreviewSize(300, 300)
12 camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
13 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
14 camRgb.setInterleaved(False)
15 camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)
16
17 # Create output
18 xoutRgb = pipeline.createXLinkOut()
19 xoutRgb.setStreamName("rgb")
20 camRgb.preview.link(xoutRgb.input)
21
22 # Pipeline defined, now the device is connected to
23 with dai.Device(pipeline) as device:
24     # Start pipeline
25     device.startPipeline()
26
27     # Output queue will be used to get the rgb frames from the output defined above
28     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
29
30     while True:
31         inRgb = qRgb.get() # blocking call, will wait until a new data has arrived
32
33         # Retrieve 'bgr' (opencv format) frame
```

(continues on next page)

(continued from previous page)

```

34         cv2.imshow("bgr", inRgb.getCvFrame())
35
36     if cv2.waitKey(1) == ord('q'):
37         break

```

We're always happy to help with code or other questions you might have.

## 3.6 02 - Mono Preview

This example shows how to set up a pipeline that outputs the left and right grayscale camera images, connects over XLink to transfer these to the host real-time, and displays both using OpenCV.

### 3.6.1 Demo

### 3.6.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

### 3.6.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  # Start defining a pipeline
7  pipeline = dai.Pipeline()
8
9  # Define a source - two mono (grayscale) cameras
10 camLeft = pipeline.createMonoCamera()
11 camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
12 camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
13
14 camRight = pipeline.createMonoCamera()
15 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
16 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
17
18 # Create outputs
19 xoutLeft = pipeline.createXLinkOut()
20 xoutLeft.setStreamName('left')
21 camLeft.out.link(xoutLeft.input)
22
23 xoutRight = pipeline.createXLinkOut()

```

(continues on next page)

(continued from previous page)

```

24 xoutRight.setStreamName('right')
25 camRight.out.link(xoutRight.input)
26
27 # Pipeline defined, now the device is connected to
28 with dai.Device(pipeline) as device:
29     # Start pipeline
30     device.startPipeline()
31
32     # Output queues will be used to get the grayscale frames from the outputs defined
33     ↪above
34     qLeft = device.getOutputQueue(name="left", maxSize=4, blocking=False)
35     qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
36
37     frameLeft = None
38     frameRight = None
39
40     while True:
41         # instead of get (blocking) used tryGet (nonblocking) which will return the
42         ↪available data or None otherwise
43         inLeft = qLeft.tryGet()
44         inRight = qRight.tryGet()
45
46         if inLeft is not None:
47             frameLeft = inLeft.getCvFrame()
48
49         if inRight is not None:
50             frameRight = inRight.getCvFrame()
51
52         # show the frames if available
53         if frameLeft is not None:
54             cv2.imshow("left", frameLeft)
55         if frameRight is not None:
56             cv2.imshow("right", frameRight)
57
58         if cv2.waitKey(1) == ord('q'):
59             break

```

We're always happy to help with code or other questions you might have.

## 3.7 03 - Depth Preview

This example shows how to set the SGBM (semi-global-matching) disparity-depth node, connects over XLink to transfer the results to the host real-time, and displays the depth map in OpenCV. Note that disparity is used in this case, as it colorizes in a more intuitive way. Below is also a preview of using different median filters side-by-side on a depth image.



### 3.7.1 Demo

Filtering depth using median filter

### 3.7.2 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

### 3.7.3 Source code

Also available on GitHub

```
1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5  import numpy as np
6
7  # Start defining a pipeline
8  pipeline = dai.Pipeline()
9
10 # Define a source - two mono (grayscale) cameras
11 left = pipeline.createMonoCamera()
12 left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
13 left.setBoardSocket(dai.CameraBoardSocket.LEFT)
14
15 right = pipeline.createMonoCamera()
16 right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
17 right.setBoardSocket(dai.CameraBoardSocket.RIGHT)
18
19 # Create a node that will produce the depth map (using disparity output as it's
20 # easier to visualize depth this way)
21 depth = pipeline.createStereoDepth()
22 depth.setConfidenceThreshold(200)
23 # Options: MEDIAN_OFF, KERNEL_3x3, KERNEL_5x5, KERNEL_7x7 (default)
24 median = dai.StereoDepthProperties.MedianFilter.KERNEL_7x7 # For depth filtering
25 depth.setMedianFilter(median)
26
27 '''
28 If one or more of the additional depth modes (lrccheck, extended, subpixel)
29 are enabled, then:
30 - depth output is FP16. TODO enable U16.
31 - median filtering is disabled on device. TODO enable.
32 - with subpixel, either depth or disparity has valid data.
33 Otherwise, depth output is U16 (mm) and median is functional.
34 But like on Gen1, either depth or disparity has valid data. TODO enable both.
35 '''
36 # Better handling for occlusions:
37 depth.setLeftRightCheck(False)
```

(continues on next page)

(continued from previous page)

```

37 # Closer-in minimum depth, disparity range is doubled:
38 depth.setExtendedDisparity(False)
39 # Better accuracy for longer distance, fractional disparity 32-levels:
40 depth.setSubpixel(False)
41
42 left.out.link(depth.left)
43 right.out.link(depth.right)
44
45 # Create output
46 xout = pipeline.createXLinkOut()
47 xout.setStreamName("disparity")
48 depth.disparity.link(xout.input)
49
50 # Pipeline defined, now the device is connected to
51 with dai.Device(pipeline) as device:
52     # Start pipeline
53     device.startPipeline()
54
55     # Output queue will be used to get the disparity frames from the outputs defined
56     ↪above
57     q = device.getOutputQueue(name="disparity", maxSize=4, blocking=False)
58
59     while True:
60         inDepth = q.get() # blocking call, will wait until a new data has arrived
61         frame = inDepth.getFrame()
62         frame = cv2.normalize(frame, None, 0, 255, cv2.NORM_MINMAX)
63         frame = cv2.applyColorMap(frame, cv2.COLORMAP_JET)
64
65         # Uncomment one of these and comment the one given above
66         # to see visualisation in different color frames
67
68         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_BONE)
69         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_AUTUMN)
70         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_WINTER)
71         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_RAINBOW)
72         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_OCEAN)
73         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_SUMMER)
74         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_SPRING)
75         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_COOL)
76         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_HSV)
77         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_HOT)
78         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_PINK)
79         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_PARULA)
80         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_MAGMA)
81         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_INFERNO)
82         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_PLASMA)
83         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_VIRIDIS)
84         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_CIVIDIS)
85         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_TWILIGHT)
86         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_TWILIGHT_SHIFTED)
87         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_TURBO)
88         #frame = cv2.applyColorMap(frame, cv2.COLORMAP_DEEPPGREEN)
89
90         # frame is ready to be shown
91         cv2.imshow("disparity", frame)
92
93         if cv2.waitKey(1) == ord('q'):

```

(continues on next page)

(continued from previous page)

93

**break**

We're always happy to help with code or other questions you might have.

## 3.8 04 - RGB Encoding

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at 8MP/4K/2160p (3840x2160) at 30FPS (the maximum possible encoding resolution possible for the encoder, higher frame-rates are possible at lower resolutions, like 1440p at 60FPS), and transfers the encoded video over XLINK to the host, saving it to disk as a video file.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave them running, you could fill up your storage on your host.

### 3.8.1 Demo

### 3.8.2 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

### 3.8.3 Source code

Also available on GitHub

```
1  #!/usr/bin/env python3
2
3  import depthai as dai
4
5  # Start defining a pipeline
6  pipeline = dai.Pipeline()
7
8  # Define a source - color camera
9  cam = pipeline.createColorCamera()
10 cam.setBoardSocket(dai.CameraBoardSocket.RGB)
11 cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
12
13 # Create an encoder, consuming the frames and encoding them using H.265 encoding
14 videoEncoder = pipeline.createVideoEncoder()
15 videoEncoder.setDefaultProfilePreset(3840, 2160, 30, dai.VideoEncoderProperties.
16   ↳ Profile.H265_MAIN)
17 cam.video.link(videoEncoder.input)
18
19 # Create output
```

(continues on next page)

(continued from previous page)

```

19 videoOut = pipeline.createXLinkOut()
20 videoOut.setStreamName('h265')
21 videoEncoder.bitstream.link(videoOut.input)
22
23 # Pipeline defined, now the device is connected to
24 with dai.Device(pipeline) as device:
25     # Start pipeline
26     device.startPipeline()
27
28     # Output queue will be used to get the encoded data from the output defined above
29     q = device.getOutputQueue(name="h265", maxSize=30, blocking=True)
30
31     # The .h265 file is a raw stream file (not playable yet)
32     with open('video.h265', 'wb') as videoFile:
33         print("Press Ctrl+C to stop encoding...")
34         try:
35             while True:
36                 h264Packet = q.get() # blocking call, will wait until a new data has
↳ arrived
37                 h264Packet.getData().tofile(videoFile) # appends the packet data to
↳ the opened file
38             except KeyboardInterrupt:
39                 # Keyboard interrupt (Ctrl + C) detected
40                 pass
41
42         print("To view the encoded data, convert the stream file (.h265) into a video
↳ file (.mp4) using a command below:")
43         print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

## 3.9 05 - RGB & Mono Encoding

This example shows how to set up the encoder node to encode the RGB camera and both grayscale cameras (of DepthAI/OAK-D) at the same time. The RGB is set to 1920x1080 and the grayscale are set to 1280x720 each, all at 30FPS. Each encoded video stream is transferred over XLINK and saved to a respective file.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave them running, you could fill up your storage on your host.

### 3.9.1 Demo

### 3.9.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

### 3.9.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import depthai as dai
4
5  # Start defining a pipeline
6  pipeline = dai.Pipeline()
7
8  # Define a source - color and mono cameras
9  colorCam = pipeline.createColorCamera()
10 monoCam = pipeline.createMonoCamera()
11 monoCam.setBoardSocket(dai.CameraBoardSocket.LEFT)
12 monoCam2 = pipeline.createMonoCamera()
13 monoCam2.setBoardSocket(dai.CameraBoardSocket.RIGHT)
14
15 # Create encoders, one for each camera, consuming the frames and encoding them using
16 ↳ H.264 / H.265 encoding
17 ve1 = pipeline.createVideoEncoder()
18 ve1.setDefaultProfilePreset(1280, 720, 30, dai.VideoEncoderProperties.Profile.H264_
19 ↳ MAIN)
20 monoCam.out.link(ve1.input)
21
22 ve2 = pipeline.createVideoEncoder()
23 ve2.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.Profile.H265_
24 ↳ MAIN)
25 colorCam.video.link(ve2.input)
26
27 ve3 = pipeline.createVideoEncoder()
28 ve3.setDefaultProfilePreset(1280, 720, 30, dai.VideoEncoderProperties.Profile.H264_
29 ↳ MAIN)
30 monoCam2.out.link(ve3.input)
31
32 # Create outputs
33 ve1Out = pipeline.createXLinkOut()
34 ve1Out.setStreamName('ve1Out')
35 ve1.bitstream.link(ve1Out.input)
36
37 ve2Out = pipeline.createXLinkOut()
38 ve2Out.setStreamName('ve2Out')
39 ve2.bitstream.link(ve2Out.input)
40
41 ve3Out = pipeline.createXLinkOut()
42 ve3Out.setStreamName('ve3Out')
43 ve3.bitstream.link(ve3Out.input)
44
45 # Pipeline defined, now the device is connected to
46 with dai.Device(pipeline) as dev:
47     # Start pipeline
48     dev.startPipeline()
49
50     # Output queues will be used to get the encoded data from the outputs defined
51     ↳ above
52     outQ1 = dev.getOutputQueue(name='ve1Out', maxSize=30, blocking=True)
53     outQ2 = dev.getOutputQueue(name='ve2Out', maxSize=30, blocking=True)

```

(continues on next page)

(continued from previous page)

```

50 outQ3 = dev.getOutputQueue(name='ve3Out', maxSize=30, blocking=True)
51
52 # The .h264 / .h265 files are raw stream files (not playable yet)
53 with open('mono1.h264', 'wb') as fileMono1H264, open('color.h265', 'wb') as
54 →fileColorH265, open('mono2.h264', 'wb') as fileMono2H264:
55     print("Press Ctrl+C to stop encoding...")
56     while True:
57         try:
58             # Empty each queue
59             while outQ1.has():
60                 outQ1.get().getData().tofile(fileMono1H264)
61
62             while outQ2.has():
63                 outQ2.get().getData().tofile(fileColorH265)
64
65             while outQ3.has():
66                 outQ3.get().getData().tofile(fileMono2H264)
67         except KeyboardInterrupt:
68             # Keyboard interrupt (Ctrl + C) detected
69             break
70
71     print("To view the encoded data, convert the stream file (.h264/.h265) into a
72 →video file (.mp4), using commands below:")
73     cmd = "ffmpeg -framerate 30 -i {} -c copy {}"
74     print(cmd.format("mono1.h264", "mono1.mp4"))
75     print(cmd.format("mono2.h264", "mono2.mp4"))
76     print(cmd.format("color.h265", "color.mp4"))

```

We're always happy to help with code or other questions you might have.

## 3.10 06 - RGB Full Resolution Saver

This example does its best to save full-resolution 3840x2160 .png files as fast as it can from the RGB sensor. It serves as an example of recording high resolution to disk for the purposes of high-resolution ground-truth data. We also recently added the options to save isp - YUV420p uncompressed frames, processed by ISP, and raw - BayerRG (R\_Gr\_Gb\_B), as read from sensor, 10-bit packed. See [here](#) for the pull request on this capability.

Be careful, this example saves full resolution .png pictures to your host storage. So if you leave them running, you could fill up your storage on your host.

### 3.10.1 Demo

### 3.10.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

### 3.10.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import time
4  from pathlib import Path
5
6  import cv2
7  import depthai as dai
8
9  # Start defining a pipeline
10 pipeline = dai.Pipeline()
11
12 # Define a source - color camera
13 camRgb = pipeline.createColorCamera()
14 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
15
16 # Create RGB output
17 xoutRgb = pipeline.createXLinkOut()
18 xoutRgb.setStreamName("rgb")
19 camRgb.video.link(xoutRgb.input)
20
21 # Create encoder to produce JPEG images
22 videoEnc = pipeline.createVideoEncoder()
23 videoEnc.setDefaultProfilePreset(camRgb.getVideoSize(), camRgb.getFps(), dai.
↳ VideoEncoderProperties.Profile.MJPEG)
24 camRgb.video.link(videoEnc.input)
25
26 # Create JPEG output
27 xoutJpeg = pipeline.createXLinkOut()
28 xoutJpeg.setStreamName("jpeg")
29 videoEnc.bitstream.link(xoutJpeg.input)
30
31
32 # Pipeline defined, now the device is connected to
33 with dai.Device(pipeline) as device:
34     # Start pipeline
35     device.startPipeline()
36
37     # Output queue will be used to get the rgb frames from the output defined above
38     qRgb = device.getOutputQueue(name="rgb", maxSize=30, blocking=False)
39     qJpeg = device.getOutputQueue(name="jpeg", maxSize=30, blocking=True)
40
41     # Make sure the destination path is present before starting to store the examples
42     Path('06_data').mkdir(parents=True, exist_ok=True)
43
44     while True:
45         inRgb = qRgb.tryGet() # non-blocking call, will return a new data that has_
↳ arrived or None otherwise
46
47         if inRgb is not None:
48             cv2.imshow("rgb", inRgb.getCvFrame())
49
50         for encFrame in qJpeg.tryGetAll():
51             with open(f"06_data/{int(time.time() * 1000)}.jpeg", "wb") as f:
52                 f.write(bytearray(encFrame.getData()))

```

(continues on next page)

(continued from previous page)

```

53
54     if cv2.waitKey(1) == ord('q'):
55         break

```

We're always happy to help with code or other questions you might have.

## 3.11 07 - Mono Full Resolution Saver

This example shows how to save 1280x720p .png of the right grayscale camera to disk. Left is defined as from the boards perspective.

### 3.11.1 Demo

### 3.11.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow *installation guide*

### 3.11.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import time
4  from pathlib import Path
5
6  import cv2
7  import depthai as dai
8
9  # Start defining a pipeline
10 pipeline = dai.Pipeline()
11
12 # Define a source - mono (grayscale) camera
13 camRight = pipeline.createMonoCamera()
14 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
15 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
16
17 # Create output
18 xoutRight = pipeline.createXLinkOut()
19 xoutRight.setStreamName("right")
20 camRight.out.link(xoutRight.input)
21
22 # Pipeline defined, now the device is connected to
23 with dai.Device(pipeline) as device:
24     # Start pipeline

```

(continues on next page)



(continued from previous page)

```

25     device.startPipeline()
26
27     # Output queue will be used to get the grayscale frames from the output defined_
↪above
28     qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
29
30     # Make sure the destination path is present before starting to store the examples
31     Path('07_data').mkdir(parents=True, exist_ok=True)
32
33     while True:
34         inRight = qRight.get() # blocking call, will wait until a new data has_
↪arrived
35         # data is originally represented as a flat 1D array, it needs to be converted_
↪into HxW form
36         frameRight = inRight.getCvFrame()
37         # frame is transformed and ready to be shown
38         cv2.imshow("right", frameRight)
39         # after showing the frame, it's being stored inside a target directory as a_
↪PNG image
40         cv2.imwrite(f"07_data/{int(time.time() * 10000)}.png", frameRight)
41
42         if cv2.waitKey(1) == ord('q'):
43             break

```

We're always happy to help with code or other questions you might have.

## 3.12 08 - RGB & MobilenetSSD

This example shows how to run MobileNetV2SSD on the RGB input frame, and how to display both the RGB preview and the metadata results from the MobileNetV2SSD on the preview.

### 3.12.1 Demo

### 3.12.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet-ssd\_openvino\_2021.2\_6shave.blob file) to work - you can download it from [here](#)

### 3.12.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import cv2
5  import depthai as dai
6  import numpy as np
7  import time
8  import argparse
9
10 nnPathDefault = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.
    ↳ 2_6shave.blob')).resolve().absolute())
11 parser = argparse.ArgumentParser()
12 parser.add_argument('nnPath', nargs='?', help="Path to mobilenet detection network_
    ↳ blob", default=nnPathDefault)
13 parser.add_argument('-s', '--sync', action="store_true", help="Sync RGB output with_
    ↳ NN output", default=False)
14 args = parser.parse_args()
15
16 # Start defining a pipeline
17 pipeline = dai.Pipeline()
18
19 # Define a source - color camera
20 camRgb = pipeline.createColorCamera()
21 camRgb.setPreviewSize(300, 300)
22 camRgb.setInterleaved(False)
23 camRgb.setFps(40)
24
25 # Define a neural network that will make predictions based on the source frames
26 nn = pipeline.createMobileNetDetectionNetwork()
27 nn.setConfidenceThreshold(0.5)
28 nn.setBlobPath(args.nnPath)
29 nn.setNumInferenceThreads(2)
30 nn.input.setBlocking(False)
31 camRgb.preview.link(nn.input)
32
33 # Create outputs
34 xoutRgb = pipeline.createXLinkOut()
35 xoutRgb.setStreamName("rgb")
36 if args.sync:
37     nn.passthrough.link(xoutRgb.input)
38 else:
39     camRgb.preview.link(xoutRgb.input)
40
41 nnOut = pipeline.createXLinkOut()
42 nnOut.setStreamName("nn")
43 nn.out.link(nnOut.input)
44
45 # MobilenetSSD label texts
46 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
    ↳ "car", "cat", "chair", "cow",
47             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
    ↳ "sheep", "sofa", "train", "tvmonitor"]
48
49

```

(continues on next page)

(continued from previous page)

```

50 # Pipeline defined, now the device is connected to
51 with dai.Device(pipeline) as device:
52     # Start pipeline
53     device.startPipeline()
54
55     # Output queues will be used to get the rgb frames and nn data from the outputs_
56     ↳defined above
57     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
58     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
59
60     startTime = time.monotonic()
61     counter = 0
62     detections = []
63     frame = None
64
65     # nn data (bounding box locations) are in <0..1> range - they need to be_
66     ↳normalized with frame width/height
67     def frameNorm(frame, bbox):
68         normVals = np.full(len(bbox), frame.shape[0])
69         normVals[::2] = frame.shape[1]
70         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
71
72     def displayFrame(name, frame):
73         for detection in detections:
74             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
75             ↳detection.ymax))
76             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
77             ↳2)
78             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
79             ↳20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
80             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
81             ↳bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
82             cv2.imshow(name, frame)
83
84     while True:
85         if args.sync:
86             # use blocking get() call to catch frame and inference result synced
87             inRgb = qRgb.get()
88             inDet = qDet.get()
89         else:
90             # instead of get (blocking) used tryGet (nonblocking) which will return_
91             ↳the available data or None otherwise
92             inRgb = qRgb.tryGet()
93             inDet = qDet.tryGet()
94
95         if inRgb is not None:
96             frame = inRgb.getCvFrame()
97             cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() -
98             ↳startTime)),
99                 (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4,
100             ↳color=(255, 255, 255))
101
102         if inDet is not None:
103             detections = inDet.detections
104             counter += 1

```

(continues on next page)

(continued from previous page)

```

98     # if the frame is available, draw bounding boxes on it and show the frame
99     if frame is not None:
100         displayFrame("rgb", frame)
101
102     if cv2.waitKey(1) == ord('q'):
103         break

```

We're always happy to help with code or other questions you might have.

## 3.13 09 - Mono & MobilenetSSD

This example shows how to run MobileNetV2SSD on the right grayscale camera and how to display the neural network results on a preview of the right camera stream.

### 3.13.1 Demo

### 3.13.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.13.3 Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14
15 # Start defining a pipeline
16 pipeline = dai.Pipeline()
17
18 # Define a source - mono (grayscale) camera
19 camRight = pipeline.createMonoCamera()

```

(continues on next page)

(continued from previous page)

```

20 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
21 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
22
23 # Define a neural network that will make predictions based on the source frames
24 nn = pipeline.createMobileNetDetectionNetwork()
25 nn.setConfidenceThreshold(0.5)
26 nn.setBlobPath(nnPath)
27 nn.setNumInferenceThreads(2)
28 nn.input.setBlocking(False)
29
30 # Create a node to convert the grayscale frame into the nn-acceptable form
31 manip = pipeline.createImageManip()
32 manip.initialConfig.setResize(300, 300)
33 # The NN model expects BGR input. By default ImageManip output type would be same as
34 # input (gray in this case)
35 manip.initialConfig.setFrameType(dai.RawImgFrame.Type.BGR888p)
36 camRight.out.link(manip.inputImage)
37 manip.out.link(nn.input)
38
39 # Create outputs
40 manipOut = pipeline.createXLinkOut()
41 manipOut.setStreamName("right")
42 manip.out.link(manipOut.input)
43
44 nnOut = pipeline.createXLinkOut()
45 nnOut.setStreamName("nn")
46 nn.out.link(nnOut.input)
47
48 # MobilenetSSD label texts
49 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
50             "car", "cat", "chair", "cow",
51             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
52             "sheep", "sofa", "train", "tvmonitor"]
53
54 # Pipeline defined, now the device is connected to
55 with dai.Device(pipeline) as device:
56     # Start pipeline
57     device.startPipeline()
58
59     # Output queues will be used to get the grayscale frames and nn data from the
60     # outputs defined above
61     qRight = device.getOutputQueue("right", maxSize=4, blocking=False)
62     qDet = device.getOutputQueue("nn", maxSize=4, blocking=False)
63
64     frame = None
65     detections = []
66
67     # nn data, being the bounding box locations, are in <0..1> range - they need to
68     # be normalized with frame width/height
69     def frameNorm(frame, bbox):
70         normVals = np.full(len(bbox), frame.shape[0])
71         normVals[::2] = frame.shape[1]
72         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
73
74     def displayFrame(name, frame):
75         for detection in detections:
76             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
77                                     detection.ymax))

```

(continues on next page)

(continued from previous page)

```

72         cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)
73         cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
74         ↪20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
75         cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
76         ↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
77         cv2.imshow(name, frame)
78
79     while True:
80         # instead of get (blocking) used tryGet (nonblocking) which will return the
81         ↪available data or None otherwise
82         inRight = qRight.tryGet()
83         inDet = qDet.tryGet()
84
85         if inRight is not None:
86             frame = inRight.getCvFrame()
87
88         if inDet is not None:
89             detections = inDet.detections
90
91         if frame is not None:
92             displayFrame("right", frame)
93
94         if cv2.waitKey(1) == ord('q'):
95             break

```

We're always happy to help with code or other questions you might have.

## 3.14 10 - Mono & MobilenetSSD & Encoding

This example shows how to run MobileNetV2SSD on the left grayscale camera in parallel with running the disparity depth results, displaying both the depth map and the right grayscale stream, with the bounding box from the neural network overlaid.

### 3.14.1 Demo

### 3.14.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.14.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14
15 # Start defining a pipeline
16 pipeline = dai.Pipeline()
17
18 # Define a source - mono (grayscale) cameras
19 left = pipeline.createMonoCamera()
20 left.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
21 left.setBoardSocket(dai.CameraBoardSocket.LEFT)
22
23 right = pipeline.createMonoCamera()
24 right.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
25 right.setBoardSocket(dai.CameraBoardSocket.RIGHT)
26
27 # Create a node that will produce the depth map (using disparity output as it's
    ↳easier to visualize depth this way)
28 stereo = pipeline.createStereoDepth()
29 stereo.setOutputRectified(True) # The rectified streams are horizontally mirrored by
    ↳default
30 stereo.setConfidenceThreshold(255)
31 stereo.setRectifyEdgeFillColor(0) # Black, to better see the cutout from
    ↳rectification (black stripe on the edges)
32
33 left.out.link(stereo.left)
34 right.out.link(stereo.right)
35
36 # Create a node to convert the grayscale frame into the nn-acceptable form
37 manip = pipeline.createImageManip()
38 manip.initialConfig.setResize(300, 300)
39 # The NN model expects BGR input. By default ImageManip output type would be same as
    ↳input (gray in this case)
40 manip.initialConfig.setFrameType(dai.RawImgFrame.Type.BGR888p)
41 stereo.rectifiedRight.link(manip.inputImage)
42
43 # Define a neural network that will make predictions based on the source frames
44 nn = pipeline.createMobileNetDetectionNetwork()
45 nn.setConfidenceThreshold(0.5)
46 nn.setBlobPath(nnPath)
47 nn.setNumInferenceThreads(2)
48 nn.input.setBlocking(False)
49 manip.out.link(nn.input)

```

(continues on next page)

(continued from previous page)

```

50
51 # Create outputs
52 depthOut = pipeline.createXLinkOut()
53 depthOut.setStreamName("depth")
54
55 stereo.disparity.link(depthOut.input)
56
57 xoutRight = pipeline.createXLinkOut()
58 xoutRight.setStreamName("rectifiedRight")
59 manip.out.link(xoutRight.input)
60
61 nnOut = pipeline.createXLinkOut()
62 nnOut.setStreamName("nn")
63 nn.out.link(nnOut.input)
64
65 # MobilenetSSD label nnLabels
66 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
67 ↪ "car", "cat", "chair", "cow",
68 ↪ "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
69 ↪ "sheep", "sofa", "train", "tvmonitor"]
70
71 # Pipeline defined, now the device is connected to
72 with dai.Device(pipeline) as device:
73     # Start pipeline
74     device.startPipeline()
75
76     # Output queues will be used to get the grayscale / depth frames and nn data from ↪
77     ↪ the outputs defined above
78     qRight = device.getOutputQueue("rectifiedRight", maxSize=4, blocking=False)
79     qDepth = device.getOutputQueue("depth", maxSize=4, blocking=False)
80     qDet = device.getOutputQueue("nn", maxSize=4, blocking=False)
81
82     rightFrame = None
83     depthFrame = None
84     detections = []
85     offsetX = (right.getResolutionWidth() - right.getResolutionHeight()) // 2
86     croppedFrame = np.zeros((right.getResolutionHeight(), right ↪
87     ↪ getResolutionHeight()))
88
89     # nn data, being the bounding box locations, are in <0..1> range - they need to ↪
90     ↪ be normalized with frame width/height
91     def frameNorm(frame, bbox):
92         normVals = np.full(len(bbox), frame.shape[0])
93         normVals[::2] = frame.shape[1]
94         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
95
96     while True:
97         # instead of get (blocking) used tryGet (nonblocking) which will return the ↪
98         ↪ available data or None otherwise
99         inRight = qRight.tryGet()
100         inDet = qDet.tryGet()
101         inDepth = qDepth.tryGet()
102
103         if inRight is not None:
104             rightFrame = inRight.getCvFrame()
105
106         if inDet is not None:

```

(continues on next page)



(continued from previous page)

```

101     detections = inDet.detections
102
103     if inDepth is not None:
104         depthFrame = cv2.flip(inDepth.getFrame(), 1)
105         # frame is transformed, the color map will be applied to highlight the_
↳depth info
106         depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_JET)
107
108         # Uncomment one of these and comment the one given above
109         # to see visualisation in different color frames
110
111         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_BONE)
112         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_AUTUMN)
113         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_WINTER)
114         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_RAINBOW)
115         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_OCEAN)
116         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_SUMMER)
117         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_SPRING)
118         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_COOL)
119         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_HSV)
120         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_HOT)
121         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_PINK)
122         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_PARULA)
123         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_MAGMA)
124         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_INFERNO)
125         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_PLASMA)
126         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_VIRIDIS)
127         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_CIVIDIS)
128         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_TWILIGHT)
129         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_TWILIGHT_
↳SHIFTED)
130         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_TURBO)
131         # depthFrame = cv2.applyColorMap(depthFrame, cv2.COLORMAP_DEEPPGREEN)
132
133     if rightFrame is not None:
134         for detection in detections:
135             bbox = frameNorm(rightFrame, (detection.xmin, detection.ymin,
↳detection.xmax, detection.ymax))
136             cv2.rectangle(rightFrame, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
↳(255, 0, 0), 2)
137             cv2.putText(rightFrame, labelMap[detection.label], (bbox[0] + 10,
↳bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
138             cv2.putText(rightFrame, f"{int(detection.confidence * 100)}%",
↳(bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
139             cv2.imshow("rectified right", rightFrame)
140
141     if depthFrame is not None:
142         for detection in detections:
143             bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin,
↳detection.xmax, detection.ymax))
144             bbox[:,2] += offsetX
145             cv2.rectangle(depthFrame, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
↳(255, 0, 0), 2)
146             cv2.putText(depthFrame, labelMap[detection.label], (bbox[0] + 10,
↳bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
147             cv2.putText(depthFrame, f"{int(detection.confidence * 100)}%",
↳(bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)

```

(continues on next page)

(continued from previous page)

```

148         cv2.imshow("depth", depthFrame)
149
150     if cv2.waitKey(1) == ord('q'):
151         break

```

We're always happy to help with code or other questions you might have.

## 3.15 11 - RGB & Encoding & Mono & MobilenetSSD

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at Full-HD resolution at 30FPS, and transfers the encoded video over XLINK to the host, saving it to disk as a video file. In the same time, a MobileNetV2SSD network is ran on the frames from right grayscale camera

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave them running, you could fill up your storage on your host.

### 3.15.1 Demo

### 3.15.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.15.3 Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳ 6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14

```

(continues on next page)

(continued from previous page)

```

15 pipeline = dai.Pipeline()
16
17 cam = pipeline.createColorCamera()
18 cam.setBoardSocket(dai.CameraBoardSocket.RGB)
19 cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
20
21 videoEncoder = pipeline.createVideoEncoder()
22 videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.
    ↳ Profile.H265_MAIN)
23 cam.video.link(videoEncoder.input)
24
25 videoOut = pipeline.createXLinkOut()
26 videoOut.setStreamName('h265')
27 videoEncoder.bitstream.link(videoOut.input)
28
29 camRight = pipeline.createMonoCamera()
30 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
31 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
32
33 nn = pipeline.createMobileNetDetectionNetwork()
34 nn.setConfidenceThreshold(0.5)
35 nn.setBlobPath(nnPath)
36 nn.setNumInferenceThreads(2)
37 nn.input.setBlocking(False)
38
39 manip = pipeline.createImageManip()
40 manip.initialConfig.setResize(300, 300)
41 # The NN model expects BGR input. By default ImageManip output type would be same as
    ↳ input (gray in this case)
42 manip.initialConfig.setFrameType(dai.RawImgFrame.Type.BGR888p)
43 camRight.out.link(manip.inputImage)
44 manip.out.link(nn.input)
45
46 xoutRight = pipeline.createXLinkOut()
47 xoutRight.setStreamName("right")
48 camRight.out.link(xoutRight.input)
49
50 manipOut = pipeline.createXLinkOut()
51 manipOut.setStreamName("manip")
52 manip.out.link(manipOut.input)
53
54 nnOut = pipeline.createXLinkOut()
55 nnOut.setStreamName("nn")
56 nn.out.link(nnOut.input)
57
58 # MobilenetSSD label texts
59 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
    ↳ "car", "cat", "chair", "cow",
60             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
    ↳ "sheep", "sofa", "train", "tvmonitor"]
61
62
63 # Pipeline defined, now the device is connected to
64 with dai.Device(pipeline) as device:
65     # Start pipeline
66     device.startPipeline()
67

```

(continues on next page)

(continued from previous page)

```

68     queue_size = 8
69     qRight = device.getOutputQueue("right", queue_size)
70     qManip = device.getOutputQueue("manip", queue_size)
71     qDet = device.getOutputQueue("nn", queue_size)
72     qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)
73
74     frame = None
75     frameManip = None
76     detections = []
77     offsetX = (camRight.getResolutionWidth() - camRight.getResolutionHeight()) // 2
78     croppedFrame = np.zeros((camRight.getResolutionHeight(), camRight.
↳ getResolutionHeight()))
79
80     def frameNorm(frame, bbox):
81         normVals = np.full(len(bbox), frame.shape[0])
82         normVals[::2] = frame.shape[1]
83         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
84
85     videoFile = open('video.h265', 'wb')
86     cv2.namedWindow("right", cv2.WINDOW_NORMAL)
87     cv2.namedWindow("manip", cv2.WINDOW_NORMAL)
88
89     while True:
90         inRight = qRight.tryGet()
91         inManip = qManip.tryGet()
92         inDet = qDet.tryGet()
93
94         while qRgbEnc.has():
95             qRgbEnc.get().getData().tofile(videoFile)
96
97         if inRight is not None:
98             frame = inRight.getCvFrame()
99
100        if inManip is not None:
101            frameManip = inManip.getCvFrame()
102
103        if inDet is not None:
104            detections = inDet.detections
105
106        if frame is not None:
107            for detection in detections:
108                bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin,
↳ detection.xmax, detection.ymax))
109                bbox[::2] += offsetX
110                cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0,
↳ 0), 2)
111                cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1]
↳ + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
112                cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] +
↳ 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
113                cv2.imshow("right", frame)
114
115        if frameManip is not None:
116            for detection in detections:
117                bbox = frameNorm(frameManip, (detection.xmin, detection.ymin,
↳ detection.xmax, detection.ymax))
118                cv2.rectangle(frameManip, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
↳ (255, 0, 0), 2)

```

(continues on next page)

(continued from previous page)

```

119         cv2.putText(frameManip, labelMap[detection.label], (bbox[0] + 10, ↵
↵bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
120         cv2.putText(frameManip, f"{int(detection.confidence * 100)}%", ↵
↵(bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
121         cv2.imshow("manip", frameManip)
122
123         if cv2.waitKey(1) == ord('q'):
124             break
125
126     videoFile.close()
127
128     print("To view the encoded data, convert the stream file (.h265) into a video ↵
↵file (.mp4) using a command below:")
129     print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

## 3.16 12 - RGB Encoding & Mono with MobilenetSSD & Depth

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at Full-HD resolution at 30FPS, and transfers the encoded video over XLINK to the host, saving it to disk as a video file. In the same time, a MobileNetV2SSD network is ran on the frames from right grayscale camera, while the application also displays the depth map produced by both of the grayscale cameras. Note that disparity is used in this case, as it colorizes in a more intuitive way.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave them running, you could fill up your storage on your host.

### 3.16.1 Demo

### 3.16.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.16.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳ 6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14
15 pipeline = dai.Pipeline()
16
17 cam = pipeline.createColorCamera()
18 cam.setBoardSocket(dai.CameraBoardSocket.RGB)
19 cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
20
21 videoEncoder = pipeline.createVideoEncoder()
22 videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.
    ↳ Profile.H265_MAIN)
23 cam.video.link(videoEncoder.input)
24
25 videoOut = pipeline.createXLinkOut()
26 videoOut.setStreamName('h265')
27 videoEncoder.bitstream.link(videoOut.input)
28 camLeft = pipeline.createMonoCamera()
29 camLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
30 camLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
31
32 camRight = pipeline.createMonoCamera()
33 camRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
34 camRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
35
36 depth = pipeline.createStereoDepth()
37 depth.setConfidenceThreshold(200)
38 # Note: the rectified streams are horizontally mirrored by default
39 depth.setOutputRectified(True)
40 depth.setRectifyEdgeFillColor(0) # Black, to better see the cutout
41 camLeft.out.link(depth.left)
42 camRight.out.link(depth.right)
43
44 depthOut = pipeline.createXLinkOut()
45 depthOut.setStreamName("depth")
46 depth.disparity.link(depthOut.input)
47
48 nn = pipeline.createMobileNetDetectionNetwork()
49 nn.setConfidenceThreshold(0.5)
50 nn.setBlobPath(nnPath)
51 nn.setNumInferenceThreads(2)
52 nn.input.setBlocking(False)

```

(continues on next page)

(continued from previous page)

```

53
54 manip = pipeline.createImageManip()
55 manip.initialConfig.setResize(300, 300)
56 # The NN model expects BGR input. By default ImageManip output type would be same as
57   ↳ input (gray in this case)
58 manip.initialConfig.setFrameType(dai.RawImgFrame.Type.BGR888p)
59 depth.rectifiedRight.link(manip.inputImage)
60 manip.out.link(nn.input)
61
62 xoutRight = pipeline.createXLinkOut()
63 xoutRight.setStreamName("right")
64 camRight.out.link(xoutRight.input)
65
66 manipOut = pipeline.createXLinkOut()
67 manipOut.setStreamName("manip")
68 manip.out.link(manipOut.input)
69
70 nnOut = pipeline.createXLinkOut()
71 nnOut.setStreamName("nn")
72 nn.out.link(nnOut.input)
73
74 # MobilenetSSD label texts
75 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
76   ↳ "car", "cat", "chair", "cow",
77   "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
78   ↳ "sheep", "sofa", "train", "tvmonitor"]
79
80 # Pipeline defined, now the device is connected to
81 with dai.Device(pipeline) as device:
82     # Start pipeline
83     device.startPipeline()
84
85     queue_size = 8
86     qRight = device.getOutputQueue("right", queue_size)
87     qDepth = device.getOutputQueue("depth", queue_size)
88     qManip = device.getOutputQueue("manip", queue_size)
89     qDet = device.getOutputQueue("nn", queue_size)
90     qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)
91
92     frame = None
93     frameManip = None
94     frameDepth = None
95     detections = []
96     offsetX = (camRight.getResolutionWidth() - camRight.getResolutionHeight()) // 2
97     croppedFrame = np.zeros((camRight.getResolutionHeight(), camRight.
98   ↳ getResolutionHeight()))
99
100     def frameNorm(frame, bbox):
101         normVals = np.full(len(bbox), frame.shape[0])
102         normVals[::2] = frame.shape[1]
103         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
104
105     videoFile = open('video.h265', 'wb')
106     cv2.namedWindow("right", cv2.WINDOW_NORMAL)
107     cv2.namedWindow("manip", cv2.WINDOW_NORMAL)
108     cv2.namedWindow("depth", cv2.WINDOW_NORMAL)

```

(continues on next page)

(continued from previous page)

```

106
107     while True:
108         inRight = qRight.tryGet()
109         inManip = qManip.tryGet()
110         inDet = qDet.tryGet()
111         inDepth = qDepth.tryGet()
112
113         while qRgbEnc.has():
114             qRgbEnc.get().getData().tofile(videoFile)
115
116         if inRight is not None:
117             frame = cv2.flip(inRight.getCvFrame(), 1)
118
119         if inManip is not None:
120             frameManip = inManip.getCvFrame()
121
122         if inDepth is not None:
123             frameDepth = cv2.flip(inDepth.getFrame(), 1)
124             frameDepth = cv2.normalize(frameDepth, None, 0, 255, cv2.NORM_MINMAX)
125             frameDepth = cv2.applyColorMap(frameDepth, cv2.COLORMAP_JET)
126
127         if inDet is not None:
128             detections = inDet.detections
129
130         if frame is not None:
131             for detection in detections:
132                 bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin,
133 ↪detection.xmax, detection.ymax))
134                 bbox[::2] += offsetX
135                 cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0,
136 ↪0), 2)
137                 cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1]
138 ↪+ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
139                 cv2.putText(frame, f"int(detection.confidence * 100) %", (bbox[0] +
140 ↪10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
141                 cv2.imshow("right", frame)
142
143         if frameDepth is not None:
144             for detection in detections:
145                 bbox = frameNorm(croppedFrame, (detection.xmin, detection.ymin,
146 ↪detection.xmax, detection.ymax))
147                 bbox[::2] += offsetX
148                 cv2.rectangle(frameDepth, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
149 ↪(255, 0, 0), 2)
150                 cv2.putText(frameDepth, labelMap[detection.label], (bbox[0] + 10,
151 ↪bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
152                 cv2.putText(frameDepth, f"int(detection.confidence * 100) %",
153 ↪(bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
154                 cv2.imshow("depth", frameDepth)
155
156         if frameManip is not None:
157             for detection in detections:
158                 bbox = frameNorm(frameManip, (detection.xmin, detection.ymin,
159 ↪detection.xmax, detection.ymax))
160                 cv2.rectangle(frameManip, (bbox[0], bbox[1]), (bbox[2], bbox[3]),
161 ↪(255, 0, 0), 2)
162                 cv2.putText(frameManip, labelMap[detection.label], (bbox[0] + 10,
163 ↪bbox[1] + 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)

```

(continues on next page)



(continued from previous page)

```

153         cv2.putText(frameManip, f" /int(detection.confidence * 100) }%",
↳ (bbox[0] + 10, bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
154         cv2.imshow("manip", frameManip)
155
156         if cv2.waitKey(1) == ord('q'):
157             break
158
159     videoFile.close()
160
161     print("To view the encoded data, convert the stream file (.h265) into a video_
↳ file (.mp4) using a command below:")
162     print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

## 3.17 13 - Encoding Max Limit

This example shows how to set up the encoder node to encode the RGB camera and both grayscale cameras (of DepthAI/OAK-D) at the same time, having all encoder parameters set to maximum quality and FPS. The RGB is set to 4K (3840x2160) and the grayscale are set to 1280x720 each, all at 25FPS. Each encoded video stream is transferred over XLINK and saved to a respective file.

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave them running, you could fill up your storage on your host.

### 3.17.1 Demo

### 3.17.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

### 3.17.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import depthai as dai
4
5  pipeline = dai.Pipeline()
6
7  # Nodes
8  colorCam = pipeline.createColorCamera()

```

(continues on next page)

(continued from previous page)

```

9 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
10 monoCam = pipeline.createMonoCamera()
11 monoCam2 = pipeline.createMonoCamera()
12 ve1 = pipeline.createVideoEncoder()
13 ve2 = pipeline.createVideoEncoder()
14 ve3 = pipeline.createVideoEncoder()
15
16 ve1Out = pipeline.createXLinkOut()
17 ve2Out = pipeline.createXLinkOut()
18 ve3Out = pipeline.createXLinkOut()
19
20 # Properties
21 monoCam.setBoardSocket(dai.CameraBoardSocket.LEFT)
22 monoCam2.setBoardSocket(dai.CameraBoardSocket.RIGHT)
23 ve1Out.setStreamName('ve1Out')
24 ve2Out.setStreamName('ve2Out')
25 ve3Out.setStreamName('ve3Out')
26
27 #setting to 26fps will trigger error
28 ve1.setDefaultProfilePreset(1280, 720, 25, dai.VideoEncoderProperties.Profile.H264_
↳MAIN)
29 ve2.setDefaultProfilePreset(3840, 2160, 25, dai.VideoEncoderProperties.Profile.H265_
↳MAIN)
30 ve3.setDefaultProfilePreset(1280, 720, 25, dai.VideoEncoderProperties.Profile.H264_
↳MAIN)
31
32 # Link nodes
33 monoCam.out.link(ve1.input)
34 colorCam.video.link(ve2.input)
35 monoCam2.out.link(ve3.input)
36
37 ve1.bitstream.link(ve1Out.input)
38 ve2.bitstream.link(ve2Out.input)
39 ve3.bitstream.link(ve3Out.input)
40
41
42 # Pipeline defined, now the device is connected to
43 with dai.Device(pipeline) as dev:
44
45     # Prepare data queues
46     outQ1 = dev.getOutputQueue('ve1Out', maxSize=30, blocking=True)
47     outQ2 = dev.getOutputQueue('ve2Out', maxSize=30, blocking=True)
48     outQ3 = dev.getOutputQueue('ve3Out', maxSize=30, blocking=True)
49
50     # Start the pipeline
51     dev.startPipeline()
52
53     # Processing loop
54     with open('mono1.h264', 'wb') as fileMono1H264, open('color.h265', 'wb') as
↳fileColorH265, open('mono2.h264', 'wb') as fileMono2H264:
55         print("Press Ctrl+C to stop encoding...")
56         while True:
57             try:
58                 # Empty each queue
59                 while outQ1.has():
60                     outQ1.get().getData().tofile(fileMono1H264)
61

```

(continues on next page)

(continued from previous page)

```

62         while outQ2.has():
63             outQ2.get().getData().tofile(fileColorH265)
64
65         while outQ3.has():
66             outQ3.get().getData().tofile(fileMonoH264)
67     except KeyboardInterrupt:
68         break
69
70     print("To view the encoded data, convert the stream file (.h264/.h265) into a
↪video file (.mp4), using commands below:")
71     cmd = "ffmpeg -framerate 25 -i {} -c copy {}"
72     print(cmd.format("mono1.h264", "mono1.mp4"))
73     print(cmd.format("mono2.h264", "mono2.mp4"))
74     print(cmd.format("color.h265", "color.mp4"))

```

We're always happy to help with code or other questions you might have.

## 3.18 14 - Color Camera Control

This example shows how to control the device-side crop and camera triggers. An output is a displayed RGB cropped frame, that can be manipulated using the following keys:

1. *a* will move the crop left
2. *d* will move the crop right
3. *w* will move the crop up
4. *s* will move the crop down
5. *c* will trigger a *still* event, causing the current frame to be captured and sent over *still* output from camera node

### 3.18.1 Demo

### 3.18.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

### 3.18.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  """
4  This example shows usage of Camera Control message as well as ColorCamera configInput,
5  ↳to change crop x and y
6  Uses 'WASD' controls to move the crop window, 'C' to capture a still image, 'T' to
7  ↳trigger autofocus, 'IOKL,.'
8  for manual exposure/focus:
9      Control:      key[dec/inc]  min..max
10     exposure time:  I   O        1..33000 [us]
11     sensitivity iso: K   L        100..1600
12     focus:          ,   .        0..255 [far..near]
13 To go back to auto controls:
14     'E' - autoexposure
15     'F' - autofocus (continuous)
16 """
17
18 import depthai as dai
19 import cv2
20
21 # Step size ('W','A','S','D' controls)
22 STEP_SIZE = 8
23 # Manual exposure/focus set step
24 EXP_STEP = 500 # us
25 ISO_STEP = 50
26 LENS_STEP = 3
27
28 pipeline = dai.Pipeline()
29
30 # Nodes
31 colorCam = pipeline.createColorCamera()
32 controlIn = pipeline.createXLinkIn()
33 configIn = pipeline.createXLinkIn()
34 videoEncoder = pipeline.createVideoEncoder()
35 stillEncoder = pipeline.createVideoEncoder()
36 videoMjpegOut = pipeline.createXLinkOut()
37 stillMjpegOut = pipeline.createXLinkOut()
38 previewOut = pipeline.createXLinkOut()
39
40 # Properties
41 colorCam.setVideoSize(640, 360)
42 colorCam.setPreviewSize(300, 300)
43 controlIn.setStreamName('control')
44 configIn.setStreamName('config')
45 videoEncoder.setDefaultProfilePreset(colorCam.getVideoSize(), colorCam.getFps(), dai.
46 ↳VideoEncoderProperties.Profile.MJPEG)
47 stillEncoder.setDefaultProfilePreset(colorCam.getStillSize(), 1, dai.
48 ↳VideoEncoderProperties.Profile.MJPEG)
49 videoMjpegOut.setStreamName('video')
50 stillMjpegOut.setStreamName('still')
51 previewOut.setStreamName('preview')

```

(continues on next page)

(continued from previous page)

```

51 # Link nodes
52 colorCam.video.link(videoEncoder.input)
53 colorCam.still.link(stillEncoder.input)
54 colorCam.preview.link(previewOut.input)
55 controlIn.out.link(colorCam.inputControl)
56 configIn.out.link(colorCam.inputConfig)
57 videoEncoder.bitstream.link(videoMjpegOut.input)
58 stillEncoder.bitstream.link(stillMjpegOut.input)
59
60
61 def clamp(num, v0, v1):
62     return max(v0, min(num, v1))
63
64
65 # Pipeline defined, now the device is connected to
66 with dai.Device(pipeline) as dev:
67
68     # Get data queues
69     controlQueue = dev.getInputQueue('control')
70     configQueue = dev.getInputQueue('config')
71     previewQueue = dev.getOutputQueue('preview')
72     videoQueue = dev.getOutputQueue('video')
73     stillQueue = dev.getOutputQueue('still')
74
75     # Start pipeline
76     dev.startPipeline()
77
78     # Max cropX & cropY
79     maxCropX = (colorCam.getResolutionWidth() - colorCam.getVideoWidth()) / colorCam.
80     ↪getResolutionWidth()
81     maxCropY = (colorCam.getResolutionHeight() - colorCam.getVideoHeight()) / ↪
82     ↪colorCam.getResolutionHeight()
83
84     # Default crop
85     cropX = 0
86     cropY = 0
87     sendCamConfig = True
88
89     # Defaults and limits for manual focus/exposure controls
90     lensPos = 150
91     lensMin = 0
92     lensMax = 255
93
94     expTime = 20000
95     expMin = 1
96     expMax = 33000
97
98     sensIso = 800
99     sensMin = 100
100     sensMax = 1600
101
102     while True:
103         previewFrames = previewQueue.tryGetAll()
104         for previewFrame in previewFrames:
105             cv2.imshow('preview', previewFrame.getData().reshape(previewFrame.
106             ↪getWidth(), previewFrame.getHeight(), 3))

```

(continues on next page)

(continued from previous page)

```

105
106     videoFrames = videoQueue.tryGetAll()
107     for videoFrame in videoFrames:
108         # Decode JPEG
109         frame = cv2.imdecode(videoFrame.getData(), cv2.IMREAD_UNCHANGED)
110         # Display
111         cv2.imshow('video', frame)
112
113         # Send new cfg to camera
114         if sendCamConfig:
115             cfg = dai.ImageManipConfig()
116             cfg.setCropRect(cropX, cropY, 0, 0)
117             configQueue.send(cfg)
118             print('Sending new crop - x: ', cropX, ' y: ', cropY)
119             sendCamConfig = False
120
121     stillFrames = stillQueue.tryGetAll()
122     for stillFrame in stillFrames:
123         # Decode JPEG
124         frame = cv2.imdecode(stillFrame.getData(), cv2.IMREAD_UNCHANGED)
125         # Display
126         cv2.imshow('still', frame)
127
128
129     # Update screen
130     key = cv2.waitKey(1)
131     if key == ord('q'):
132         break
133     elif key == ord('c'):
134         ctrl = dai.CameraControl()
135         ctrl.setCaptureStill(True)
136         controlQueue.send(ctrl)
137     elif key == ord('t'):
138         print("Autofocus trigger (and disable continuous)")
139         ctrl = dai.CameraControl()
140         ctrl.setAutoFocusMode(dai.CameraControl.AutoFocusMode.AUTO)
141         ctrl.setAutoFocusTrigger()
142         controlQueue.send(ctrl)
143     elif key == ord('f'):
144         print("Autofocus enable, continuous")
145         ctrl = dai.CameraControl()
146         ctrl.setAutoFocusMode(dai.CameraControl.AutoFocusMode.CONTINUOUS_VIDEO)
147         controlQueue.send(ctrl)
148     elif key == ord('e'):
149         print("Autoexposure enable")
150         ctrl = dai.CameraControl()
151         ctrl.setAutoExposureEnable()
152         controlQueue.send(ctrl)
153     elif key in [ord(','), ord('.')]:
154         if key == ord(','): lensPos -= LENS_STEP
155         if key == ord('.'): lensPos += LENS_STEP
156         lensPos = clamp(lensPos, lensMin, lensMax)
157         print("Setting manual focus, lens position:", lensPos)
158         ctrl = dai.CameraControl()
159         ctrl.setManualFocus(lensPos)
160         controlQueue.send(ctrl)
161     elif key in [ord('i'), ord('o'), ord('k'), ord('l')]:

```

(continues on next page)

(continued from previous page)

```

162     if key == ord('i'): expTime -= EXP_STEP
163     if key == ord('o'): expTime += EXP_STEP
164     if key == ord('k'): sensIso -= ISO_STEP
165     if key == ord('l'): sensIso += ISO_STEP
166     expTime = clamp(expTime, expMin, expMax)
167     sensIso = clamp(sensIso, sensMin, sensMax)
168     print("Setting manual exposure, time:", expTime, "iso:", sensIso)
169     ctrl = dai.CameraControl()
170     ctrl.setManualExposure(expTime, sensIso)
171     controlQueue.send(ctrl)
172     elif key in [ord('w'), ord('a'), ord('s'), ord('d')]:
173         if key == ord('a'):
174             cropX = cropX - (maxCropX / colorCam.getResolutionWidth()) * STEP_SIZE
175             if cropX < 0: cropX = maxCropX
176         elif key == ord('d'):
177             cropX = cropX + (maxCropX / colorCam.getResolutionWidth()) * STEP_SIZE
178             if cropX > maxCropX: cropX = 0
179         elif key == ord('w'):
180             cropY = cropY - (maxCropY / colorCam.getResolutionHeight()) * STEP_
181             if cropY < 0: cropY = maxCropY
182         elif key == ord('s'):
183             cropY = cropY + (maxCropY / colorCam.getResolutionHeight()) * STEP_
184             if cropY > maxCropY: cropY = 0
185         sendCamConfig = True

```

We're always happy to help with code or other questions you might have.

## 3.19 15 - 4K RGB MobileNetSSD

This example shows how to MobileNetv2SSD on the RGB input frame, and how to display both the RGB preview and the metadata results from the MobileNetv2SSD on the preview. The preview size is set to 4K resolution

### 3.19.1 Demo

### 3.19.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.19.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳5shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14 # Start defining a pipeline
15 pipeline = dai.Pipeline()
16
17 # Define a source - color camera
18 camRgb = pipeline.createColorCamera()
19 camRgb.setPreviewSize(300, 300)    # NN input
20 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_4_K)
21 camRgb.setInterleaved(False)
22 camRgb.setPreviewKeepAspectRatio(False)
23
24 # Define a neural network that will make predictions based on the source frames
25 nn = pipeline.createMobileNetDetectionNetwork()
26 nn.setConfidenceThreshold(0.5)
27 nn.setBlobPath(nnPath)
28 nn.setNumInferenceThreads(2)
29 nn.input.setBlocking(False)
30 camRgb.preview.link(nn.input)
31
32 # Create outputs
33 xoutVideo = pipeline.createXLinkOut()
34 xoutVideo.setStreamName("video")
35 camRgb.video.link(xoutVideo.input)
36
37 xoutPreview = pipeline.createXLinkOut()
38 xoutPreview.setStreamName("preview")
39 camRgb.preview.link(xoutPreview.input)
40
41 nnOut = pipeline.createXLinkOut()
42 nnOut.setStreamName("nn")
43 nn.out.link(nnOut.input)
44
45 # MobilenetSSD label texts
46 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
    ↳"car", "cat", "chair", "cow",
47             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
    ↳"sheep", "sofa", "train", "tvmonitor"]
48
49 # Pipeline defined, now the device is connected to
50 with dai.Device(pipeline) as device:
51     # Start pipeline

```

(continues on next page)



(continued from previous page)

```

52     device.startPipeline()
53
54     # Output queues will be used to get the frames and nn data from the outputs_
55     ↳ defined above
56     qVideo = device.getOutputQueue(name="video", maxSize=4, blocking=False)
57     qPreview = device.getOutputQueue(name="preview", maxSize=4, blocking=False)
58     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
59
60     previewFrame = None
61     videoFrame = None
62     detections = []
63
64     # nn data, being the bounding box locations, are in <0..1> range - they need to_
65     ↳ be normalized with frame width/height
66     def frameNorm(frame, bbox):
67         normVals = np.full(len(bbox), frame.shape[0])
68         normVals[::2] = frame.shape[1]
69         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
70
71     def displayFrame(name, frame):
72         for detection in detections:
73             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
74             ↳ detection.ymax))
75             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
76             ↳ 2)
77             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
78             ↳ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
79             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
80             ↳ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
81             cv2.imshow(name, frame)
82
83     cv2.namedWindow("video", cv2.WINDOW_NORMAL)
84     cv2.resizeWindow("video", 1280, 720)
85     print("Resize video window with mouse drag!")
86
87     while True:
88         # instead of get (blocking) used tryGet (nonblocking) which will return the_
89         ↳ available data or None otherwise
90         inVideo = qVideo.tryGet()
91         inPreview = qPreview.tryGet()
92         inDet = qDet.tryGet()
93
94         if inVideo is not None:
95             videoFrame = inVideo.getCvFrame()
96
97         if inPreview is not None:
98             previewFrame = inPreview.getCvFrame()
99
100         if inDet is not None:
101             detections = inDet.detections
102
103         if videoFrame is not None:
104             displayFrame("video", videoFrame)
105
106         if previewFrame is not None:
107             displayFrame("preview", previewFrame)

```

(continues on next page)

(continued from previous page)

```

102     if cv2.waitKey(1) == ord('q'):
103         break

```

We're always happy to help with code or other questions you might have.

## 3.20 16 - Device Queue Event

This example shows how to use `getQueueEvent` function in order to be notified when one of the packets from selected streams arrive

### 3.20.1 Demo

### 3.20.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

### 3.20.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  # This example demonstrates use of queue events to block a thread until a message
4  # arrives to any (of the specified) queue
5
6  import cv2
7  import depthai as dai
8
9  # Start defining a pipeline
10 pipeline = dai.Pipeline()
11
12 # Create Color and Mono cameras
13 camRgb = pipeline.createColorCamera()
14 camMono = pipeline.createMonoCamera()
15 # Create separate streams for them
16 xoutRgb = pipeline.createXLinkOut()
17 xoutMono = pipeline.createXLinkOut()
18
19 # Set properties
20 xoutRgb.setStreamName("rgb")
21 xoutMono.setStreamName("mono")
22 # Cap color camera to 5 fps
23 camRgb.setFps(5)
24 camRgb.setInterleaved(True)
25 camRgb.setPreviewSize(300, 300)

```

(continues on next page)

(continued from previous page)

```

26
27 # Connect
28 camRgb.preview.link(xoutRgb.input)
29 camMono.out.link(xoutMono.input)
30
31
32 # Pipeline defined, now the device is connected to
33 with dai.Device(pipeline) as device:
34     # Start pipeline
35     device.startPipeline()
36
37     # Clear queue events
38     device.getQueueEvents()
39
40     while True:
41         # Block until a message arrives to any of the specified queues
42         queueName = device.getQueueEvent(("rgb", "mono"))
43
44         # Getting that message from queue with name specified by the event
45         # Note: number of events doesn't necessarily match number of messages in
↳ queues
46         # because queues can be set to non-blocking (overwriting) behavior
47         message = device.getOutputQueue(queueName).get()
48
49         # display arrived frames
50         if type(message) == dai.ImgFrame:
51             cv2.imshow(queueName, message.getCvFrame())
52
53         if cv2.waitKey(1) == ord('q'):
54             break

```

We're always happy to help with code or other questions you might have.

## 3.21 17 - Video & MobilenetSSD

This example shows how to MobileNetv2SSD on the RGB input frame, which is read from the specified file, and not from the RGB camera, and how to display both the RGB frame and the metadata results from the MobileNetv2SSD on the frame. DepthAI is used here only as a processing unit

### 3.21.1 Demo

### 3.21.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) and prerecorded video (construction\_vest.mp4 file) to work - you can download them here: [mobilenet.blob](#) and [construction\\_vest.mp4](#)

### 3.21.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  from time import monotonic
9
10 # Get argument first
11 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳8shave.blob')).resolve().absolute())
12 videoPath = str(Path("./construction_vest.mp4").resolve().absolute())
13 if len(sys.argv) > 2:
14     nnPath = sys.argv[1]
15     videoPath = sys.argv[2]
16
17 # Start defining a pipeline
18 pipeline = dai.Pipeline()
19
20
21 # Create neural network input
22 xinDet = pipeline.createXLinkIn()
23 xinDet.setStreamName("inDet")
24
25 # Define a neural network that will make predictions based on the source frames
26 nn = pipeline.createMobileNetDetectionNetwork()
27 nn.setConfidenceThreshold(0.5)
28 nn.setBlobPath(nnPath)
29 nn.setNumInferenceThreads(2)
30 nn.input.setBlocking(False)
31 xinDet.out.link(nn.input)
32
33 # Create output
34 nnOut = pipeline.createXLinkOut()
35 nnOut.setStreamName("nn")
36 nn.out.link(nnOut.input)
37
38 # MobilenetSSD label texts
39 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
    ↳"car", "cat", "chair", "cow",
40             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
    ↳"sheep", "sofa", "train", "tvmonitor"]
41
42
43 # Pipeline defined, now the device is connected to
44 with dai.Device(pipeline) as device:
45     # Start pipeline
46     device.startPipeline()
47
48     # Output queues will be used to get the rgb frames and nn data from the outputs_
    ↳defined above
49     qIn = device.getInputQueue(name="inDet")
50     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)

```

(continues on next page)

(continued from previous page)

```

51     frame = None
52     detections = []
53
54     # nn data, being the bounding box locations, are in <0..1> range - they need to
55     ↪be normalized with frame width/height
56     def frameNorm(frame, bbox):
57         normVals = np.full(len(bbox), frame.shape[0])
58         normVals[::2] = frame.shape[1]
59         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
60
61
62     def to_planar(arr: np.ndarray, shape: tuple) -> np.ndarray:
63         return cv2.resize(arr, shape).transpose(2, 0, 1).flatten()
64
65     def displayFrame(name, frame):
66         for detection in detections:
67             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
68             ↪detection.ymax))
69             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
70             ↪2)
71             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
72             ↪20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
73             cv2.putText(frame, f"/int(detection.confidence * 100) %", (bbox[0] + 10,
74             ↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
75             cv2.imshow(name, frame)
76
77     cap = cv2.VideoCapture(videoPath)
78     while cap.isOpened():
79         read_correctly, frame = cap.read()
80         if not read_correctly:
81             break
82
83         img = dai.ImgFrame()
84         img.setData(to_planar(frame, (300, 300)))
85         img.setTimestamp(monotonic())
86         img.setWidth(300)
87         img.setHeight(300)
88         qIn.send(img)
89
90         inDet = qDet.tryGet()
91
92         if inDet is not None:
93             detections = inDet.detections
94
95         if frame is not None:
96             displayFrame("rgb", frame)
97
98         if cv2.waitKey(1) == ord('q'):
99             break

```

We're always happy to help with code or other questions you might have.

## 3.22 18 - RGB Encoding with MobilenetSSD

This example shows how to configure the depthai video encoder in h.265 format to encode the RGB camera input at Full-HD resolution at 30FPS, and transfers the encoded video over XLINK to the host, saving it to disk as a video file. In the same time, a MobileNetv2SSD network is ran on the frames from the same RGB camera that is used for encoding

Pressing Ctrl+C will stop the recording and then convert it using ffmpeg into an mp4 to make it playable. Note that ffmpeg will need to be installed and runnable for the conversion to mp4 to succeed.

Be careful, this example saves encoded video to your host storage. So if you leave them running, you could fill up your storage on your host.

### 3.22.1 Demo

### 3.22.2 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.22.3 Source code

Also available on [GitHub](#)

```
1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Get argument first
10 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
    ↳6shave.blob')).resolve().absolute())
11 if len(sys.argv) > 1:
12     nnPath = sys.argv[1]
13
14
15 pipeline = dai.Pipeline()
16
17 cam = pipeline.createColorCamera()
18 cam.setBoardSocket(dai.CameraBoardSocket.RGB)
19 cam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
20 cam.setPreviewSize(300, 300)
21 cam.setInterleaved(False)
22
23 videoEncoder = pipeline.createVideoEncoder()
```

(continues on next page)

(continued from previous page)

```

24 videoEncoder.setDefaultProfilePreset(1920, 1080, 30, dai.VideoEncoderProperties.
    ↪ Profile.H265_MAIN)
25 cam.video.link(videoEncoder.input)
26
27 nn = pipeline.createMobileNetDetectionNetwork()
28 nn.setConfidenceThreshold(0.5)
29 nn.setBlobPath(nnPath)
30 nn.setNumInferenceThreads(2)
31 nn.input.setBlocking(False)
32 cam.preview.link(nn.input)
33
34 videoOut = pipeline.createXLinkOut()
35 videoOut.setStreamName('h265')
36 videoEncoder.bitstream.link(videoOut.input)
37
38 xoutRgb = pipeline.createXLinkOut()
39 xoutRgb.setStreamName("rgb")
40 cam.preview.link(xoutRgb.input)
41
42 nnOut = pipeline.createXLinkOut()
43 nnOut.setStreamName("nn")
44 nn.out.link(nnOut.input)
45
46 # MobilenetSSD label texts
47 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
    ↪ "car", "cat", "chair", "cow",
48             "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
    ↪ "sheep", "sofa", "train", "tvmonitor"]
49
50 with dai.Device(pipeline) as device, open('video.h265', 'wb') as videoFile:
51     device.startPipeline()
52
53     queue_size = 8
54     qRgb = device.getOutputQueue("rgb", queue_size)
55     qDet = device.getOutputQueue("nn", queue_size)
56     qRgbEnc = device.getOutputQueue('h265', maxSize=30, blocking=True)
57
58     frame = None
59     detections = []
60
61
62     def frameNorm(frame, bbox):
63         normVals = np.full(len(bbox), frame.shape[0])
64         normVals[::2] = frame.shape[1]
65         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
66
67     def displayFrame(name, frame):
68         for detection in detections:
69             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
    ↪ detection.ymax))
70             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
    ↪ 2)
71             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
    ↪ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
72             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
    ↪ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
73             cv2.imshow(name, frame)

```

(continues on next page)

(continued from previous page)

```

74
75
76     while True:
77         inRgb = qRgb.tryGet()
78         inDet = qDet.tryGet()
79
80         while qRgbEnc.has():
81             qRgbEnc.get().getData().tofile(videoFile)
82
83         if inRgb is not None:
84             frame = inRgb.getCvFrame()
85
86         if inDet is not None:
87             detections = inDet.detections
88
89         if frame is not None:
90             displayFrame("rgb", frame)
91
92         if cv2.waitKey(1) == ord('q'):
93             break
94
95 print("To view the encoded data, convert the stream file (.h265) into a video file (.
96 ↪mp4) using a command below:")
97 print("ffmpeg -framerate 30 -i video.h265 -c copy video.mp4")

```

We're always happy to help with code or other questions you might have.

## 3.23 21 - RGB & MobilenetSSD decoding on device

This example shows how to run MobileNetV2SSD on the RGB input frame, and how to display both the RGB preview and the metadata results from the MobileNetV2SSD on the preview. It's similar to example '08\_rgb\_mobilenet' except decoding is done on Myriad instead on the host.

setConfidenceThreshold - confidence threshold above which objects are detected

### 3.23.1 Demo

### 3.23.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)



### 3.23.3 Source code

Also available on [GitHub](#)

```
1 print("Migrated to 08_rgb_mobilenet.py")
```

We're always happy to help with code or other questions you might have.

## 3.24 22.1 - RGB & TinyYoloV3 decoding on device

This example shows how to run TinyYoloV3 on the RGB input frame, and how to display both the RGB preview and the metadata results from the TinyYoloV3 on the preview. Decoding is done on Myriad instead on the host.

Configurable, network dependent parameters are required for correct decoding: setNumClasses - number of YOLO classes setCoordinateSize - size of coordinate setAnchors - yolo anchors setAnchorMasks - anchorMasks26, anchorMasks13 (anchorMasks52 - additionally for full YOLOv3) setIouThreshold - intersection over union threshold setConfidenceThreshold - confidence threshold above which objects are detected

### 3.24.1 Demo

### 3.24.2 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.24.3 Source code

Also available on [GitHub](#)

```
1 #!/usr/bin/env python3
2
3 """
4 Tiny-yolo-v3 device side decoding demo
5 YOLO v3 Tiny is a real-time object detection model implemented with Keras* from
6 this repository <https://github.com/david8862/keras-YOLOv3-model-set> and converted
7 to TensorFlow* framework. This model was pretrained on COCO* dataset with 80
8 classes.
9 """
10 from pathlib import Path
11 import sys
12 import cv2
13 import depthai as dai
14 import numpy as np
15 import time
```

(continues on next page)

(continued from previous page)

```

17 # tiny yolo v3 label texts
18 labelMap = [
19     "person",      "bicycle",    "car",            "motorbike",     "aeroplane",
20     ↪ "bus",        "train",          "traffic light",  "fire hydrant",  "stop sign",
21     ↪ "truck",      "boat",          "parking meter", "bench",         "bird",
22     ↪ "cat",        "dog",           "horse",         "sheep",
23     ↪ "cow",        "elephant",      "giraffe",       "backpack",      "umbrella",
24     ↪ "bear",      "zebra",         "handbag",       "tie",           "suitcase",
25     ↪ "frisbee",   "skis",          "snowboard",     "sports ball",
26     ↪ "kite",      "baseball bat",  "baseball glove", "skateboard",    "surfboard",
27     ↪ "tennis racket", "bottle",
28     ↪ "wine glass", "cup",           "fork",          "knife",         "spoon",
29     ↪ "bowl",      "banana",
30     ↪ "apple",     "sandwich",     "orange",        "broccoli",     "carrot",
31     ↪ "hot dog",   "pizza",
32     ↪ "donut",     "cake",         "chair",         "sofa",          "pottedplant",
33     ↪ "bed",       "diningtable",
34     ↪ "toilet",    "tvmonitor",    "laptop",        "mouse",         "remote",
35     ↪ "keyboard", "cell phone",
36     ↪ "microwave", "oven",         "toaster",       "sink",          "refrigerator",
37     ↪ "book",      "clock",
38     ↪ "vase",      "scissors",     "teddy bear",    "hair drier",   "toothbrush"
39 ]
40
41 syncNN = True
42
43 # Get argument first
44 nnPath = str((Path(__file__).parent / Path('models/tiny-yolo-v3_openvino_2021.2_
45 ↪ 6shave.blob')).resolve().absolute())
46
47 if len(sys.argv) > 1:
48     nnPath = sys.argv[1]
49
50 # Start defining a pipeline
51 pipeline = dai.Pipeline()
52
53 # Define a source - color camera
54 camRgb = pipeline.createColorCamera()
55 camRgb.setPreviewSize(416, 416)
56 camRgb.setInterleaved(False)
57 camRgb.setFps(40)
58
59 # network specific settings
60 detectionNetwork = pipeline.createYoloDetectionNetwork()
61 detectionNetwork.setConfidenceThreshold(0.5)
62 detectionNetwork.setNumClasses(80)
63 detectionNetwork.setCoordinateSize(4)
64 detectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
65 detectionNetwork.setAnchorMasks({"side26": np.array([1, 2, 3]), "side13": np.array([3,
66 ↪ 4, 5])})
67 detectionNetwork.setIouThreshold(0.5)
68
69 detectionNetwork.setBlobPath(nnPath)
70 detectionNetwork.setNumInferenceThreads(2)

```

(continues on next page)

(continued from previous page)

```

61 detectionNetwork.input.setBlocking(False)
62
63 camRgb.preview.link(detectionNetwork.input)
64
65 # Create outputs
66 xoutRgb = pipeline.createXLinkOut()
67 xoutRgb.setStreamName("rgb")
68 if syncNN:
69     detectionNetwork.passthrough.link(xoutRgb.input)
70 else:
71     camRgb.preview.link(xoutRgb.input)
72
73 nnOut = pipeline.createXLinkOut()
74 nnOut.setStreamName("detections")
75 detectionNetwork.out.link(nnOut.input)
76
77
78 # Pipeline defined, now the device is connected to
79 with dai.Device(pipeline) as device:
80     # Start pipeline
81     device.startPipeline()
82
83     # Output queues will be used to get the rgb frames and nn data from the outputs,
84     ↳ defined above
85     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
86     qDet = device.getOutputQueue(name="detections", maxSize=4, blocking=False)
87
88     frame = None
89     detections = []
90
91     # nn data, being the bounding box locations, are in <0..1> range - they need to
92     ↳ be normalized with frame width/height
93     def frameNorm(frame, bbox):
94         normVals = np.full(len(bbox), frame.shape[0])
95         normVals[::2] = frame.shape[1]
96         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
97
98     def displayFrame(name, frame):
99         for detection in detections:
100             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
101             ↳ detection.ymax))
102             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
103             ↳ 2)
104             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
105             ↳ 20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
106             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
107             ↳ bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
108             cv2.imshow(name, frame)
109
110     startTime = time.monotonic()
111     counter = 0
112
113     while True:
114         if syncNN:
115             inRgb = qRgb.get()
116             inDet = qDet.get()
117         else:

```

(continues on next page)

(continued from previous page)

```

112         inRgb = qRgb.tryGet()
113         inDet = qDet.tryGet()
114
115         if inRgb is not None:
116             frame = inRgb.getCvFrame()
117             cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() -
118 ↪startTime)),
119                             (2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4,
120 ↪color=(255, 255, 255))
121
122         if inDet is not None:
123             detections = inDet.detections
124             counter += 1
125
126         if frame is not None:
127             displayFrame("rgb", frame)
128
129         if cv2.waitKey(1) == ord('q'):
130             break

```

We're always happy to help with code or other questions you might have.

## 3.25 22.2 - RGB & TinyYoloV4 decoding on device

This example shows how to run TinyYoloV4 on the RGB input frame, and how to display both the RGB preview and the metadata results from the TinyYoloV4 on the preview. Decoding is done on Myriad instead on the host.

Configurable, network dependent parameters are required for correct decoding: setNumClasses - number of YOLO classes setCoordinateSize - size of coordinate setAnchors - yolo anchors setAnchorMasks - anchorMasks26, anchorMasks13 (anchorMasks52 - additionally for full YOLOv4) setIouThreshold - intersection over union threshold setConfidenceThreshold - confidence threshold above which objects are detected

### 3.25.1 Demo

### 3.25.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires YOLOv4-tiny blob (tiny-yolo-v4\_openvino\_2021.2\_6shave.blob file) to work - you can download it from [here](#)

### 3.25.3 Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  """
4  Tiny-yolo-v4 device side decoding demo
5  The code is the same as for Tiny-yolo-V3, the only difference is the blob file.
6  The blob was compiled following this tutorial: https://github.com/TNTWEN/OpenVINO-
   ↳ YOLOV4
7  """
8
9  from pathlib import Path
10 import sys
11 import cv2
12 import depthai as dai
13 import numpy as np
14 import time
15
16 # tiny yolo v4 label texts
17 labelMap = [
18     "person",          "bicycle",    "car",          "motorbike",    "aeroplane",
19     ↳ "bus",            "train",    "truck",        "boat",        "traffic light", "fire hydrant", "stop sign",
20     ↳ "parking meter", "bench",    "bird",         "cat",         "dog",          "horse",        "sheep",
21     ↳ "cow",            "elephant", "bear",         "zebra",       "giraffe",      "backpack",     "umbrella",
22     ↳ "handbag",       "tie",      "suitcase",     "frisbee",     "skis",         "snowboard",   "sports ball",
23     ↳ "kite",          "baseball bat", "baseball glove", "skateboard", "surfboard",   "tennis racket", "bottle",
24     ↳ "wine glass",    "cup",      "fork",         "knife",       "spoon",        "bowl",         "banana",
25     ↳ "apple",         "sandwich", "orange",       "broccoli",   "carrot",       "hot dog",      "pizza",
26     ↳ "donut",        "cake",     "chair",        "sofa",       "pottedplant",  "bed",          "diningtable",
27     ↳ "toilet",       "tvmonitor", "laptop",      "mouse",      "remote",       "keyboard",     "cell phone",
28     ↳ "microwave",    "oven",     "toaster",     "sink",       "refrigerator", "book",         "clock",
29     ↳ "vase",         "scissors", "teddy bear",  "hair drier", "toothbrush"
30 ]
31
32
33 syncNN = True
34
35 # Get argument first
36 nnPath = str((Path(__file__).parent / Path('models/tiny-yolo-v4_openvino_2021.2_
   ↳ 6shave.blob')).resolve().absolute())
37 if len(sys.argv) > 1:
38     nnPath = sys.argv[1]
39
40 # Start defining a pipeline
41 pipeline = dai.Pipeline()

```

(continues on next page)

(continued from previous page)

```

42
43 # Define a source - color camera
44 camRgb = pipeline.createColorCamera()
45 camRgb.setPreviewSize(416, 416)
46 camRgb.setInterleaved(False)
47 camRgb.setFps(40)
48
49 # network specific settings
50 detectionNetwork = pipeline.createYoloDetectionNetwork()
51 detectionNetwork.setConfidenceThreshold(0.5)
52 detectionNetwork.setNumClasses(80)
53 detectionNetwork.setCoordinateSize(4)
54 detectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,319]))
55 detectionNetwork.setAnchorMasks({"side26": np.array([1, 2, 3]), "side13": np.array([3,
56     ↳ 4, 5])})
57 detectionNetwork.setIouThreshold(0.5)
58
59 detectionNetwork.setBlobPath(nnPath)
60 detectionNetwork.setNumInferenceThreads(2)
61 detectionNetwork.input.setBlocking(False)
62
63 camRgb.preview.link(detectionNetwork.input)
64
65 # Create outputs
66 xoutRgb = pipeline.createXLinkOut()
67 xoutRgb.setStreamName("rgb")
68 if syncNN:
69     detectionNetwork.passthrough.link(xoutRgb.input)
70 else:
71     camRgb.preview.link(xoutRgb.input)
72
73 nnOut = pipeline.createXLinkOut()
74 nnOut.setStreamName("detections")
75 detectionNetwork.out.link(nnOut.input)
76
77 # Pipeline defined, now the device is connected to
78 with dai.Device(pipeline) as device:
79     # Start pipeline
80     device.startPipeline()
81
82     # Output queues will be used to get the rgb frames and nn data from the outputs_
83     ↳defined above
84     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
85     qDet = device.getOutputQueue(name="detections", maxSize=4, blocking=False)
86
87     frame = None
88     detections = []
89
90     # nn data, being the bounding box locations, are in <0..1> range - they need to_
91     ↳be normalized with frame width/height
92     def frameNorm(frame, bbox):
93         normVals = np.full(len(bbox), frame.shape[0])
94         normVals[::2] = frame.shape[1]
95         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
96
97     def displayFrame(name, frame):

```

(continues on next page)

(continued from previous page)

```

96         for detection in detections:
97             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
↪detection.ymax))
98             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
↪2)
99             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
↪20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
100             cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
101             cv2.imshow(name, frame)
102
103     startTime = time.monotonic()
104     counter = 0
105
106     while True:
107         if syncNN:
108             inRgb = qRgb.get()
109             inDet = qDet.get()
110         else:
111             inRgb = qRgb.tryGet()
112             inDet = qDet.tryGet()
113
114         if inRgb is not None:
115             frame = inRgb.getCvFrame()
116             cv2.putText(frame, "NN fps: {:.2f}".format(counter / (time.monotonic() -
↪startTime)),
117             ↪(2, frame.shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4,
↪color=(255, 255, 255))
118
119         if inDet is not None:
120             detections = inDet.detections
121             counter += 1
122
123         if frame is not None:
124             displayFrame("rgb", frame)
125
126         if cv2.waitKey(1) == ord('q'):
127             break

```

We're always happy to help with code or other questions you might have.

## 3.26 23 - Auto Exposure on ROI

This example shows how to dynamically set the Auto Exposure (AE) of the RGB camera dynamically, during application runtime, based on bounding box position

### 3.26.1 Demo

### 3.26.2 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.26.3 Usage

By default, AutoExposure region is adjusted based on neural network output. If desired, the region can be set manually. You can do so by pressing one of the following buttons:

- *w* - move AE region up
- *s* - move AE region down
- *a* - move AE region left
- *d* - move AE region right
- *n* - deactivate manual region (switch back to nn-based roi)

### 3.26.4 Source code

Also available on [GitHub](#)

```
1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8
9  # Press WASD to move a manual ROI window for auto-exposure control.
10 # Press N to go back to the region controlled by the NN detections.
11
12 # Get argument first
13 nnPath = str((Path(__file__).parent / Path('models/mobilenet-ssd_openvino_2021.2_
14 ↪5shave.blob')).resolve().absolute())
15 if len(sys.argv) > 1:
16     nnPath = sys.argv[1]
17
18 previewSize = (300, 300)
19
20 # Start defining a pipeline
21 pipeline = dai.Pipeline()
22
23 # Define a source - color camera
24 camRgb = pipeline.createColorCamera()
```

(continues on next page)



(continued from previous page)

```

24 camRgb.setPreviewSize(*previewSize)
25 camRgb.setInterleaved(False)
26
27 camControlIn = pipeline.createXLinkIn()
28 camControlIn.setStreamName('camControl')
29 camControlIn.out.link(camRgb.inputControl)
30
31 # Define a neural network that will make predictions based on the source frames
32 nn = pipeline.createMobileNetDetectionNetwork()
33 nn.setConfidenceThreshold(0.5)
34 nn.setBlobPath(nnPath)
35 nn.setNumInferenceThreads(2)
36 nn.input.setBlocking(False)
37 camRgb.preview.link(nn.input)
38
39 # Create outputs
40 xoutRgb = pipeline.createXLinkOut()
41 xoutRgb.setStreamName("rgb")
42 camRgb.preview.link(xoutRgb.input)
43
44 nnOut = pipeline.createXLinkOut()
45 nnOut.setStreamName("nn")
46 nn.out.link(nnOut.input)
47
48 # MobilenetSSD label texts
49 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
50 ↪ "car", "cat", "chair", "cow",
51 ↪ "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
52 ↪ "sheep", "sofa", "train", "tvmonitor"]
53
54 def clamp(num, v0, v1):
55     return max(v0, min(num, v1))
56
57 def asControl(roi):
58     camControl = dai.CameraControl()
59     camControl.setAutoExposureRegion(*roi)
60     return camControl
61
62
63 class AutoExposureRegion:
64     step = 10
65     position = (0, 0)
66     size = (100, 100)
67     resolution = camRgb.getResolutionSize()
68     maxDims = previewSize[0], previewSize[1]
69
70     def grow(self, x=0, y=0):
71         self.size = (
72             clamp(x + self.size[0], 1, self.maxDims[0]),
73             clamp(y + self.size[1], 1, self.maxDims[1])
74         )
75
76     def move(self, x=0, y=0):
77         self.position = (
78             clamp(x + self.position[0], 0, self.maxDims[0]),

```

(continues on next page)

(continued from previous page)

```

79         clamp(y + self.position[1], 0, self.maxDims[1])
80     )
81
82     def endPosition(self):
83         return (
84             clamp(self.position[0] + self.size[0], 0, self.maxDims[0]),
85             clamp(self.position[1] + self.size[1], 0, self.maxDims[1]),
86         )
87
88     def toRoi(self):
89         roi = np.array([*self.position, *self.size])
90         # Convert to absolute camera coordinates
91         roi = roi * self.resolution[1] // 300
92         roi[0] += (self.resolution[0] - self.resolution[1]) // 2 # x offset for
93         device crop
94         return roi
95
96     @staticmethod
97     def bboxToRoi(bbox):
98         startX, startY = bbox[:2]
99         width, height = bbox[2] - startX, bbox[3] - startY
100         roi = frameNorm(np.empty(camRgb.getResolutionSize()), (startX, startY, width,
101         height))
102         return roi
103
104 # Pipeline defined, now the device is connected to
105 with dai.Device(pipeline) as device:
106     # Start pipeline
107     device.startPipeline()
108
109     # Output queues will be used to get the rgb frames and nn data from the outputs
110     defined above
111     qControl = device.getInputQueue(name="camControl")
112     qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
113     qDet = device.getOutputQueue(name="nn", maxSize=4, blocking=False)
114     frame = None
115     detections = []
116
117     nnRegion = True
118     region = AutoExposureRegion()
119
120     # nn data (bounding box locations) are in <0..1> range - they need to be
121     normalized with frame width/height
122     def frameNorm(frame, bbox):
123         normVals = np.full(len(bbox), frame.shape[0])
124         normVals[::2] = frame.shape[1]
125         return (np.clip(np.array(bbox), 0, 1) * normVals).astype(int)
126
127     def displayFrame(name, frame):
128         for detection in detections:
129             bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax,
130             detection.ymax))
131             cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0),
132             2)
133             cv2.putText(frame, labelMap[detection.label], (bbox[0] + 10, bbox[1] +
134             20), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)

```

(continues on next page)

(continued from previous page)

```

129         cv2.putText(frame, f"{int(detection.confidence * 100)}%", (bbox[0] + 10,
↪bbox[1] + 40), cv2.FONT_HERSHEY_TRIPLEX, 0.5, 255)
130         if not nnRegion:
131             cv2.rectangle(frame, region.position, region.endPosition(), (0, 255, 0),
↪2)
132         cv2.imshow(name, frame)
133
134     while True:
135         # instead of get (blocking) used tryGet (nonblocking) which will return the
↪available data or None otherwise
136         inRgb = qRgb.tryGet()
137         inDet = qDet.tryGet()
138
139         if inRgb is not None:
140             frame = inRgb.getCvFrame()
141
142         if inDet is not None:
143             detections = inDet.detections
144
145             if nnRegion and len(detections) > 0:
146                 bbox = (detections[0].xmin, detections[0].ymin, detections[0].xmax,
↪detections[0].ymax)
147                 qControl.send(asControl(AutoExposureRegion.bboxToRoi(bbox)))
148
149         if frame is not None:
150             displayFrame("rgb", frame)
151
152         key = cv2.waitKey(1)
153         if key == ord('n'):
154             print("AE ROI controlled by NN")
155             nnRegion = True
156         elif key in [ord('w'), ord('a'), ord('s'), ord('d'), ord('+'), ord('-')]:
157             nnRegion = False
158             if key == ord('a'):
159                 region.move(x=-region.step)
160             if key == ord('d'):
161                 region.move(x=region.step)
162             if key == ord('w'):
163                 region.move(y=-region.step)
164             if key == ord('s'):
165                 region.move(y=region.step)
166             if key == ord('+'):
167                 region.grow(x=10, y=10)
168                 region.step = region.step + 1
169             if key == ord('-'):
170                 region.grow(x=-10, y=-10)
171                 region.step = max(region.step - 1, 1)
172             print(f"Setting static AE ROI: {region.toRoi()} (on frame: {[*region.
↪position, *region.endPosition()]})")
173             qControl.send(asControl(region.toRoi()))
174         elif key == ord('q'):
175             break

```

We're always happy to help with code or other questions you might have.

## 3.27 24 - OpenCV support

This example shows API which exposes both numpy and OpenCV compatible image types for easier usage. It uses ColorCamera node to retrieve both BGR interleaved 'preview' and NV12 encoded 'video' frames. Both are displayed using functions `getFrame` and `getCvFrame`.

### 3.27.1 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.27.2 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  # Start defining a pipeline
7  pipeline = dai.Pipeline()
8
9  # Define a source - color camera
10 camRgb = pipeline.createColorCamera()
11 camRgb.setPreviewSize(300, 300)
12 camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
13 camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
14 camRgb.setInterleaved(True)
15 camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
16
17 # Create output
18 xoutVideo = pipeline.createXLinkOut()
19 xoutVideo.setStreamName("video")
20 xoutPreview = pipeline.createXLinkOut()
21 xoutPreview.setStreamName("preview")
22
23 camRgb.preview.link(xoutPreview.input)
24 camRgb.video.link(xoutVideo.input)
25
26 # Pipeline defined, now the device is connected to
27 with dai.Device(pipeline) as device:
28     # Start pipeline
29     device.startPipeline()
30
31     while True:
32         # Get preview and video frames
33         preview = device.getOutputQueue('preview').get()
```

(continues on next page)

(continued from previous page)

```

34     video = device.getOutputQueue('video').get()
35
36     # Show 'preview' frame as is (already in correct format, no copy is made)
37     cv2.imshow("preview", preview.getFrame())
38     # Get BGR frame from NV12 encoded video frame to show with opencv
39     cv2.imshow("video", video.getCvFrame())
40
41     if cv2.waitKey(1) == ord('q'):
42         break

```

We're always happy to help with code or other questions you might have.

## 3.28 25 - System information

This example shows how to get system information (memory usage, cpu usage and temperature) from the board.

### 3.28.1 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

For additional information, please follow [Python API installation guide](#)

### 3.28.2 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6
7  def print_sys_info(info):
8      m = 1024 * 1024 # MiB
9      print(f"Drr used / total - {info._DDRMemoryUsage.used / m:.2f} / {info.
10 ↪DDRMemoryUsage.total / m:.2f} MiB")
11      print(f"Cmx used / total - {info.cmxMemoryUsage.used / m:.2f} / {info.
12 ↪cmxMemoryUsage.total / m:.2f} MiB")
13      print(f"LeonCss heap used / total - {info.leonCssMemoryUsage.used / m:.2f} /
14 ↪{info.leonCssMemoryUsage.total / m:.2f} MiB")
15      print(f"LeonMss heap used / total - {info.leonMssMemoryUsage.used / m:.2f} /
16 ↪{info.leonMssMemoryUsage.total / m:.2f} MiB")
17      t = info.chipTemperature
18      print(f"Chip temperature - average: {t.average:.2f}, css: {t.css:.2f}, mss: {t.
19 ↪mss:.2f}, upa0: {t.upa:.2f}, upa1: {t.dss:.2f}")
20      print(f"Cpu usage - Leon OS: {info.leonCssCpuUsage.average * 100:.2f}%, Leon RT:
21 ↪{info.leonMssCpuUsage.average * 100:.2f} %")

```

(continues on next page)

(continued from previous page)

```

16     print("-----")
17
18
19 # Start defining a pipeline
20 pipeline = dai.Pipeline()
21
22 sys_logger = pipeline.createSystemLogger()
23 sys_logger.setRate(1) # 1 Hz
24
25 # Create output
26 linkOut = pipeline.createXLinkOut()
27 linkOut.setStreamName("sysinfo")
28 sys_logger.out.link(linkOut.input)
29
30 # Pipeline defined, now the device is connected to
31 with dai.Device(pipeline) as device:
32     # Start pipeline
33     device.startPipeline()
34
35     # Output queue will be used to get the system info
36     q_sysinfo = device.getOutputQueue(name="sysinfo", maxSize=4, blocking=False)
37
38     while True:
39         info = q_sysinfo.get() # blocking call, will wait until a new data has
40         ↪ arrived
41         print_sys_info(info)
42
43         if cv2.waitKey(1) == ord('q'):
44             break

```

We're always happy to help with code or other questions you might have.

## 3.29 26.1 - RGB & MobilenetSSD with spatial data

This example shows how to run MobileNetV2SSD on the RGB input frame, and how to display both the RGB preview, detections, depth map and spatial information (X,Y,Z). It's similar to example '21\_mobilenet\_decoding\_on\_device' except it has spatial data. X,Y,Z coordinates are relative to the center of depth map.

setConfidenceThreshold - confidence threshold above which objects are detected

### 3.29.1 Demo

### 3.29.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.29.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  import time
9
10 '''
11 Spatial detection network demo.
12 Performs inference on RGB camera and retrieves spatial location coordinates: x,y,
13 ↪z relative to the center of depth map.
14 '''
15
16 # MobilenetSSD label texts
17 labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
18 ↪"car", "cat", "chair", "cow",
19 ↪"diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
20 ↪"sheep", "sofa", "train", "tvmonitor"]
21
22 syncNN = True
23
24 # Get argument first
25 nnBlobPath = str((Path(__file__).parent / Path('models/mobilenet.blob')).resolve().
26 ↪absolute())
27
28 if len(sys.argv) > 1:
29     nnBlobPath = sys.argv[1]
30
31 # Start defining a pipeline
32 pipeline = dai.Pipeline()
33
34 # Define a source - color camera
35 colorCam = pipeline.createColorCamera()
36 spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
37 monoLeft = pipeline.createMonoCamera()
38 monoRight = pipeline.createMonoCamera()
39 stereo = pipeline.createStereoDepth()
40
41 xoutRgb = pipeline.createXLinkOut()
42 xoutNN = pipeline.createXLinkOut()
43 xoutBoundingBoxDepthMapping = pipeline.createXLinkOut()
44 xoutDepth = pipeline.createXLinkOut()
45
46 xoutRgb.setStreamName("rgb")
47 xoutNN.setStreamName("detections")
48 xoutBoundingBoxDepthMapping.setStreamName("boundingBoxDepthMapping")
49 xoutDepth.setStreamName("depth")
50
51 colorCam.setPreviewSize(300, 300)
52 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
53 colorCam.setInterleaved(False)
54 colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)

```

(continues on next page)

(continued from previous page)

```

51
52 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
53 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
54 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
55 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
56
57 # setting node configs
58 stereo.setOutputDepth(True)
59 stereo.setConfidenceThreshold(255)
60
61 spatialDetectionNetwork.setBlobPath(nnBlobPath)
62 spatialDetectionNetwork.setConfidenceThreshold(0.5)
63 spatialDetectionNetwork.input.setBlocking(False)
64 spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
65 spatialDetectionNetwork.setDepthLowerThreshold(100)
66 spatialDetectionNetwork.setDepthUpperThreshold(5000)
67
68 # Create outputs
69
70 monoLeft.out.link(stereo.left)
71 monoRight.out.link(stereo.right)
72
73 colorCam.preview.link(spatialDetectionNetwork.input)
74 if(syncNN):
75     spatialDetectionNetwork.passthrough.link(xoutRgb.input)
76 else:
77     colorCam.preview.link(xoutRgb.input)
78
79 spatialDetectionNetwork.out.link(xoutNN.input)
80 spatialDetectionNetwork.boundingBoxMapping.link(xoutBoundingBoxDepthMapping.input)
81
82 stereo.depth.link(spatialDetectionNetwork.inputDepth)
83 spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)
84
85 # Pipeline defined, now the device is connected to
86 with dai.Device(pipeline) as device:
87     # Start pipeline
88     device.startPipeline()
89
90     # Output queues will be used to get the rgb frames and nn data from the outputs_
91     ↪defined above
92     previewQueue = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
93     detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4,
94     ↪blocking=False)
95     xoutBoundingBoxDepthMapping = device.getOutputQueue(name="boundingBoxDepthMapping
96     ↪", maxSize=4, blocking=False)
97     depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
98
99     frame = None
100     detections = []
101
102     startTime = time.monotonic()
103     counter = 0
104     fps = 0
105     color = (255, 255, 255)
106
107     while True:

```

(continues on next page)



(continued from previous page)

```

105     inPreview = previewQueue.get()
106     inNN = detectionNNQueue.get()
107     depth = depthQueue.get()
108
109     counter+=1
110     current_time = time.monotonic()
111     if (current_time - startTime) > 1 :
112         fps = counter / (current_time - startTime)
113         counter = 0
114         startTime = current_time
115
116     frame = inPreview.getCvFrame()
117     depthFrame = depth.getFrame()
118
119     depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↳CV_8UC1)
120     depthFrameColor = cv2.equalizeHist(depthFrameColor)
121     depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
122     detections = inNN.detections
123     if len(detections) != 0:
124         boundingBoxMapping = xoutBoundingBoxDepthMapping.get()
125         roiDatas = boundingBoxMapping.getConfigData()
126
127         for roiData in roiDatas:
128             roi = roiData.roi
129             roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
↳shape[0])
130             topLeft = roi.topLeft()
131             bottomRight = roi.bottomRight()
132             xmin = int(topLeft.x)
133             ymin = int(topLeft.y)
134             xmax = int(bottomRight.x)
135             ymax = int(bottomRight.y)
136
137             cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↳FONT_HERSHEY_SCRIPT_SIMPLEX)
138
139
140     # if the frame is available, draw bounding boxes on it and show the frame
141     height = frame.shape[0]
142     width = frame.shape[1]
143     for detection in detections:
144         # denormalize bounding box
145         x1 = int(detection.xmin * width)
146         x2 = int(detection.xmax * width)
147         y1 = int(detection.ymin * height)
148         y2 = int(detection.ymax * height)
149         try:
150             label = labelMap[detection.label]
151         except:
152             label = detection.label
153         cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↳TRIPLEX, 0.5, color)
154         cv2.putText(frame, "{:.2f}".format(detection.confidence*100), (x1 + 10,
↳y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
155         cv2.putText(frame, f"X: {int(detection.spatialCoordinates.x)} mm", (x1 +
↳10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)

```

(continues on next page)

(continued from previous page)

```

156         cv2.putText(frame, f"Y: {int(detection.spatialCoordinates.y)} mm", (x1 +
↪10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
157         cv2.putText(frame, f"Z: {int(detection.spatialCoordinates.z)} mm", (x1 +
↪10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
158
159         cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)
160
161         cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.
↪FONT_HERSHEY_TRIPLEX, 0.4, color)
162         cv2.imshow("depth", depthFrameColor)
163         cv2.imshow("rgb", frame)
164
165         if cv2.waitKey(1) == ord('q'):
166             break

```

We're always happy to help with code or other questions you might have.

## 3.30 26.2 - MONO & MobilenetSSD with spatial data

This example shows how to run MobileNetV2SSD on the rectified right input frame, and how to display both the preview, detections, depth map and spatial information (X,Y,Z). It's similar to example '21\_mobilenet\_decoding\_on\_device' except it has spatial data. X,Y,Z coordinates are relative to the center of depth map.

setConfidenceThreshold - confidence threshold above which objects are detected

### 3.30.1 Demo

### 3.30.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSSD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.30.3 Source code

Also available on GitHub

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  import time
9

```

(continues on next page)

(continued from previous page)

```

10  '''
11  Mobilenet SSD device side decoding demo
12  The "mobilenet-ssd" model is a Single-Shot multibox Detection (SSD) network intended
13  to perform object detection. This model is implemented using the Caffe* framework.
14  For details about this model, check out the repository <https://github.com/
    ↪chuanqi305/MobileNet-SSD>.
15  '''
16
17  # MobilenetSSD label texts
18  labelMap = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus",
    ↪"car", "cat", "chair", "cow",
19              "diningtable", "dog", "horse", "motorbike", "person", "pottedplant",
    ↪"sheep", "sofa", "train", "tvmonitor"]
20
21  syncNN = True
22  flipRectified = True
23
24  # Get argument first
25  nnPath = str((Path(__file__).parent / Path('models/mobilenet.blob')).resolve()).
    ↪absolute())
26  if len(sys.argv) > 1:
27      nnPath = sys.argv[1]
28
29  # Start defining a pipeline
30  pipeline = dai.Pipeline()
31
32
33  manip = pipeline.createImageManip()
34  manip.initialConfig.setResize(300, 300)
35  # The NN model expects BGR input. By default ImageManip output type would be same as
    ↪input (gray in this case)
36  manip.initialConfig setFrameType(dai.RawImgFrame.Type.BGR888p)
37  # manip.setKeepAspectRatio(False)
38
39  # Define a neural network that will make predictions based on the source frames
40  spatialDetectionNetwork = pipeline.createMobileNetSpatialDetectionNetwork()
41  spatialDetectionNetwork.setConfidenceThreshold(0.5)
42  spatialDetectionNetwork.setBlobPath(nnPath)
43  spatialDetectionNetwork.input.setBlocking(False)
44  spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
45  spatialDetectionNetwork.setDepthLowerThreshold(100)
46  spatialDetectionNetwork.setDepthUpperThreshold(5000)
47
48  manip.out.link(spatialDetectionNetwork.input)
49
50  # Create outputs
51  xoutManip = pipeline.createXLinkOut()
52  xoutManip.setStreamName("right")
53  if(syncNN):
54      spatialDetectionNetwork.passthrough.link(xoutManip.input)
55  else:
56      manip.out.link(xoutManip.input)
57
58  depthRoiMap = pipeline.createXLinkOut()
59  depthRoiMap.setStreamName("boundingBoxDepthMapping")
60
61  xoutDepth = pipeline.createXLinkOut()

```

(continues on next page)

(continued from previous page)

```

62 xoutDepth.setStreamName("depth")
63
64 nnOut = pipeline.createXLinkOut()
65 nnOut.setStreamName("detections")
66 spatialDetectionNetwork.out.link(nnOut.input)
67 spatialDetectionNetwork.boundingBoxMapping.link(depthRoiMap.input)
68
69 monoLeft = pipeline.createMonoCamera()
70 monoRight = pipeline.createMonoCamera()
71 stereo = pipeline.createStereoDepth()
72 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
73 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
74 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
75 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
76 stereo.setOutputDepth(True)
77 stereo.setConfidenceThreshold(255)
78 stereo.setOutputRectified(True)
79
80 stereo.rectifiedRight.link(manip.inputImage)
81
82 monoLeft.out.link(stereo.left)
83 monoRight.out.link(stereo.right)
84
85 stereo.depth.link(spatialDetectionNetwork.inputDepth)
86 spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)
87
88 # Pipeline defined, now the device is connected to
89 with dai.Device(pipeline) as device:
90     # Start pipeline
91     device.startPipeline()
92
93     # Output queues will be used to get the rgb frames and nn data from the outputs_
94     ↳ defined above
95     previewQueue = device.getOutputQueue(name="right", maxSize=4, blocking=False)
96     detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4,
97     ↳ blocking=False)
98     depthRoiMap = device.getOutputQueue(name="boundingBoxDepthMapping", maxSize=4,
99     ↳ blocking=False)
100     depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
101
102     rectifiedRight = None
103     detections = []
104
105     startTime = time.monotonic()
106     counter = 0
107     fps = 0
108     color = (255, 255, 255)
109
110     while True:
111         inRectified = previewQueue.get()
112         det = detectionNNQueue.get()
113         depth = depthQueue.get()
114
115         counter += 1
116         currentTime = time.monotonic()
117         if (currentTime - startTime) > 1:
118             fps = counter / (currentTime - startTime)

```

(continues on next page)

(continued from previous page)

```

116         counter = 0
117         startTime = currentTime
118
119         rectifiedRight = inRectified.getCvFrame()
120
121         depthFrame = depth.getFrame()
122
123         depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
↪CV_8UC1)
124         depthFrameColor = cv2.equalizeHist(depthFrameColor)
125         depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
126         detections = det.detections
127         if len(detections) != 0:
128             boundingBoxMapping = depthRoiMap.get()
129             roiDatas = boundingBoxMapping.getConfigData()
130
131             for roiData in roiDatas:
132                 roi = roiData.roi
133                 roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
↪shape[0])
134                 topLeft = roi.topLeft()
135                 bottomRight = roi.bottomRight()
136                 xmin = int(topLeft.x)
137                 ymin = int(topLeft.y)
138                 xmax = int(bottomRight.x)
139                 ymax = int(bottomRight.y)
140                 cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↪FONT_HERSHEY_SCRIPT_SIMPLEX)
141
142                 if flipRectified:
143                     rectifiedRight = cv2.flip(rectifiedRight, 1)
144
145                 # if the rectifiedRight is available, draw bounding boxes on it and show the
↪rectifiedRight
146                 height = rectifiedRight.shape[0]
147                 width = rectifiedRight.shape[1]
148                 for detection in detections:
149                     if flipRectified:
150                         swap = detection.xmin
151                         detection.xmin = 1 - detection.xmax
152                         detection.xmax = 1 - swap
153                     # denormalize bounding box
154                     x1 = int(detection.xmin * width)
155                     x2 = int(detection.xmax * width)
156                     y1 = int(detection.ymin * height)
157                     y2 = int(detection.ymax * height)
158
159                     try:
160                         label = labelMap[detection.label]
161                     except:
162                         label = detection.label
163
164                     cv2.putText(rectifiedRight, str(label), (x1 + 10, y1 + 20), cv2.FONT_
↪HERSHEY_TRIPLEX, 0.5, color)
165                     cv2.putText(rectifiedRight, "{:.2f}".format(detection.confidence*100),
↪
↪(x1 + 10, y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
166                     cv2.putText(rectifiedRight, f"X: {int(detection.spatialCoordinates.x)} mm
↪", (x1 + 10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)

```

(continues on next page)

(continued from previous page)

```

167         cv2.putText(rectifiedRight, f"Y: {int(detection.spatialCoordinates.y)} mm
↪", (x1 + 10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
168         cv2.putText(rectifiedRight, f"Z: {int(detection.spatialCoordinates.z)} mm
↪", (x1 + 10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
169
170         cv2.rectangle(rectifiedRight, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_
↪SIMPLEX)
171
172         cv2.putText(rectifiedRight, "NN fps: {:.2f}".format(fps), (2, rectifiedRight.
↪shape[0] - 4), cv2.FONT_HERSHEY_TRIPLEX, 0.4, color)
173         cv2.imshow("depth", depthFrameColor)
174         cv2.imshow("rectified right", rectifiedRight)
175
176         if cv2.waitKey(1) == ord('q'):
177             break

```

We're always happy to help with code or other questions you might have.

## 3.31 26.1 - RGB & TinyYolo with spatial data

This example shows how to run TinyYoloV3 and v4 on the RGB input frame, and how to display both the RGB preview, detections, depth map and spatial information (X,Y,Z). It's similar to example '26\_1\_spatial\_mobilenet' except it is running TinyYolo network. X,Y,Z coordinates are relative to the center of depth map.

setNumClasses - number of YOLO classes setCoordinateSize - size of coordinate setAnchors - yolo anchors setAnchorMasks - anchorMasks26, anchorMasks13 (anchorMasks52 - additionally for full YOLOv4) setIouThreshold - intersection over union threshold setConfidenceThreshold - confidence threshold above which objects are detected

### 3.31.1 Demo

### 3.31.2 Setup

Please run the following command to install the required dependencies

```

python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai

```

For additional information, please follow [installation guide](#)

This example also requires YOLOv4-tiny blob (tiny-yolo-v4\_openvino\_2021.2\_6shave.blob file) to work - you can download it from [here](#)

YOLOv3-tiny blob (tiny-yolo-v3\_openvino\_2021.2\_6shave.blob file) can be used too - you can download it from [here](#)

### 3.31.3 Source code

Also available on [GitHub](#)

```

1  #!/usr/bin/env python3
2
3  from pathlib import Path
4  import sys
5  import cv2
6  import depthai as dai
7  import numpy as np
8  import time
9
10 '''
11 Spatial Tiny-yolo example
12 Performs inference on RGB camera and retrieves spatial location coordinates: x,y,z_
13 ↳relative to the center of depth map.
14 Can be used for tiny-yolo-v3 or tiny-yolo-v4 networks
15 '''
16
17 # tiny yolo v3/4 label texts
18 labelMap = [
19     "person",      "bicycle",    "car",            "motorbike",     "aeroplane",
20     ↳"bus",         "train",      "traffic light",  "fire hydrant",  "stop sign",
21     ↳"truck",      "boat",       "parking meter", "bench",         "dog",          "horse",         "sheep",
22     ↳"bird",       "cat",        "cow",           "elephant",      "giraffe",      "backpack",      "umbrella",
23     ↳"bear",       "zebra",      "handbag",       "tie",           "skis",         "snowboard",     "sports ball",
24     ↳"suitcase",   "frisbee",    "baseball glove", "baseball bat",  "surfboard",    "tennis racket", "bottle",
25     ↳"kite",       "skateboard", "wine glass",    "cup",           "fork",         "knife",         "spoon",
26     ↳"apple",      "sandwich",   "baseball",      "glove",         "bowl",         "banana",
27     ↳"orange",    "broccoli",   "carrot",        "hot dog",       "pizza",
28     ↳"donut",     "cake",       "chair",         "sofa",          "pottedplant",  "bed",           "diningtable",
29     ↳"toilet",    "tvmonitor",  "laptop",        "mouse",         "remote",       "keyboard",      "cell phone",
30     ↳"microwave", "oven",       "toaster",       "sink",          "refrigerator", "book",          "clock",
31     ↳"vase",      "scissors",   "teddy bear",    "hair drier",    "toothbrush"
32 ]
33
34 syncNN = True
35
36 # Get argument first
37 nnBlobPath = str((Path(__file__).parent / Path('models/mobilenet.blob')).resolve()).
38 ↳absolute())
39
40 if len(sys.argv) > 1:
41     nnBlobPath = sys.argv[1]
42
43 # Start defining a pipeline
44 pipeline = dai.Pipeline()
45

```

(continues on next page)

(continued from previous page)

```

42 # Define a source - color camera
43 colorCam = pipeline.createColorCamera()
44 spatialDetectionNetwork = pipeline.createYoloSpatialDetectionNetwork()
45 monoLeft = pipeline.createMonoCamera()
46 monoRight = pipeline.createMonoCamera()
47 stereo = pipeline.createStereoDepth()
48
49 xoutRgb = pipeline.createXLinkOut()
50 xoutNN = pipeline.createXLinkOut()
51 xoutBoundingBoxDepthMapping = pipeline.createXLinkOut()
52 xoutDepth = pipeline.createXLinkOut()
53
54 xoutRgb.setStreamName("rgb")
55 xoutNN.setStreamName("detections")
56 xoutBoundingBoxDepthMapping.setStreamName("boundingBoxDepthMapping")
57 xoutDepth.setStreamName("depth")
58
59
60 colorCam.setPreviewSize(416, 416)
61 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
62 colorCam.setInterleaved(False)
63 colorCam.setColorOrder(dai.ColorCameraProperties.ColorOrder.BGR)
64
65 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
66 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
67 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
68 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
69
70 # setting node configs
71 stereo.setOutputDepth(True)
72 stereo.setConfidenceThreshold(255)
73
74 spatialDetectionNetwork.setBlobPath(nnBlobPath)
75 spatialDetectionNetwork.setConfidenceThreshold(0.5)
76 spatialDetectionNetwork.input.setBlocking(False)
77 spatialDetectionNetwork.setBoundingBoxScaleFactor(0.5)
78 spatialDetectionNetwork.setDepthLowerThreshold(100)
79 spatialDetectionNetwork.setDepthUpperThreshold(5000)
80 # yolo specific parameters
81 spatialDetectionNetwork.setNumClasses(80)
82 spatialDetectionNetwork.setCoordinateSize(4)
83 spatialDetectionNetwork.setAnchors(np.array([10,14, 23,27, 37,58, 81,82, 135,169, 344,
84 ↪ 319]))
85 spatialDetectionNetwork.setAnchorMasks({ "side26": np.array([1,2,3]), "side13": np.
86 ↪ array([3,4,5]) })
87 spatialDetectionNetwork.setIouThreshold(0.5)
88
89 # Create outputs
90 monoLeft.out.link(stereo.left)
91 monoRight.out.link(stereo.right)
92
93 colorCam.preview.link(spatialDetectionNetwork.input)
94 if(syncNN):
95     spatialDetectionNetwork.passthrough.link(xoutRgb.input)
96 else:
97     colorCam.preview.link(xoutRgb.input)

```

(continues on next page)



(continued from previous page)

```

97
98 spatialDetectionNetwork.out.link(xoutNN.input)
99 spatialDetectionNetwork.boundingBoxMapping.link(xoutBoundingBoxDepthMapping.input)
100
101 stereo.depth.link(spatialDetectionNetwork.inputDepth)
102 spatialDetectionNetwork.passthroughDepth.link(xoutDepth.input)
103
104 # Pipeline defined, now the device is connected to
105 with dai.Device(pipeline) as device:
106     # Start pipeline
107     device.startPipeline()
108
109     # Output queues will be used to get the rgb frames and nn data from the outputs_
110     ↪defined above
111     previewQueue = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
112     detectionNNQueue = device.getOutputQueue(name="detections", maxSize=4, ↪
113     ↪blocking=False)
114     xoutBoundingBoxDepthMapping = device.getOutputQueue(name="boundingBoxDepthMapping
115     ↪", maxSize=4, blocking=False)
116     depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
117
118     frame = None
119     detections = []
120
121     startTime = time.monotonic()
122     counter = 0
123     fps = 0
124     color = (255, 255, 255)
125
126     while True:
127         inPreview = previewQueue.get()
128         inNN = detectionNNQueue.get()
129         depth = depthQueue.get()
130
131         counter+=1
132         current_time = time.monotonic()
133         if (current_time - startTime) > 1 :
134             fps = counter / (current_time - startTime)
135             counter = 0
136             startTime = current_time
137
138         frame = inPreview.getCvFrame()
139         depthFrame = depth.getFrame()
140
141         depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.
142         ↪CV_8UC1)
143         depthFrameColor = cv2.equalizeHist(depthFrameColor)
144         depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
145         detections = inNN.detections
146         if len(detections) != 0:
147             boundingBoxMapping = xoutBoundingBoxDepthMapping.get()
148             roiDatas = boundingBoxMapping.getConfigData()
149
150             for roiData in roiDatas:
151                 roi = roiData.roi
152                 roi = roi.denormalize(depthFrameColor.shape[1], depthFrameColor.
153                 ↪shape[0])

```

(continues on next page)

(continued from previous page)

```

149         topLeft = roi.topLeft()
150         bottomRight = roi.bottomRight()
151         xmin = int(topLeft.x)
152         ymin = int(topLeft.y)
153         xmax = int(bottomRight.x)
154         ymax = int(bottomRight.y)
155
156         cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.
↪FONT_HERSHEY_SCRIPT_SIMPLEX)
157
158
159         # if the frame is available, draw bounding boxes on it and show the frame
160         height = frame.shape[0]
161         width = frame.shape[1]
162         for detection in detections:
163             # denormalize bounding box
164             x1 = int(detection.xmin * width)
165             x2 = int(detection.xmax * width)
166             y1 = int(detection.ymin * height)
167             y2 = int(detection.ymax * height)
168             try:
169                 label = labelMap[detection.label]
170             except:
171                 label = detection.label
172             cv2.putText(frame, str(label), (x1 + 10, y1 + 20), cv2.FONT_HERSHEY_
↪TRIPLEX, 0.5, color)
173             cv2.putText(frame, "{:.2f}".format(detection.confidence*100), (x1 + 10,
↪y1 + 35), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
174             cv2.putText(frame, f"X: {int(detection.spatialCoordinates.x)} mm", (x1 +
↪10, y1 + 50), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
175             cv2.putText(frame, f"Y: {int(detection.spatialCoordinates.y)} mm", (x1 +
↪10, y1 + 65), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
176             cv2.putText(frame, f"Z: {int(detection.spatialCoordinates.z)} mm", (x1 +
↪10, y1 + 80), cv2.FONT_HERSHEY_TRIPLEX, 0.5, color)
177
178             cv2.rectangle(frame, (x1, y1), (x2, y2), color, cv2.FONT_HERSHEY_SIMPLEX)
179
180             cv2.putText(frame, "NN fps: {:.2f}".format(fps), (2, frame.shape[0] - 4), cv2.
↪FONT_HERSHEY_TRIPLEX, 0.4, color)
181             cv2.imshow("depth", depthFrameColor)
182             cv2.imshow("rgb", frame)
183
184             if cv2.waitKey(1) == ord('q'):
185                 break

```

We're always happy to help with code or other questions you might have.

## 3.32 27 - Spatial location calculator

This example shows how to retrieve spatial location data (X,Y,Z) on a runtime configurable ROI. X,Y,Z coordinates are relative to the center of depth map.

setConfidenceThreshold - confidence threshold above which objects are detected

### 3.32.1 Demo

### 3.32.2 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow [installation guide](#)

This example also requires MobilenetSDD blob (mobilenet.blob file) to work - you can download it from [here](#)

### 3.32.3 Source code

Also available on [GitHub](#)

```
1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5
6  stepSize = 0.05
7
8  # Start defining a pipeline
9  pipeline = dai.Pipeline()
10
11 # Define a source - two mono (grayscale) cameras
12 monoLeft = pipeline.createMonoCamera()
13 monoRight = pipeline.createMonoCamera()
14 stereo = pipeline.createStereoDepth()
15 spatialLocationCalculator = pipeline.createSpatialLocationCalculator()
16
17 xoutDepth = pipeline.createXLinkOut()
18 xoutSpatialData = pipeline.createXLinkOut()
19 xinSpatialCalcConfig = pipeline.createXLinkIn()
20
21 xoutDepth.setStreamName("depth")
22 xoutSpatialData.setStreamName("spatialData")
23 xinSpatialCalcConfig.setStreamName("spatialCalcConfig")
24
25 # MonoCamera
26 monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
27 monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
28 monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
29 monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
30
```

(continues on next page)

(continued from previous page)

```

31 outputDepth = True
32 outputRectified = False
33 lrcheck = False
34 subpixel = False
35
36 # StereoDepth
37 stereo.setOutputDepth(outputDepth)
38 stereo.setOutputRectified(outputRectified)
39 stereo.setConfidenceThreshold(255)
40
41 stereo.setLeftRightCheck(lrcheck)
42 stereo.setSubpixel(subpixel)
43
44 monoLeft.out.link(stereo.left)
45 monoRight.out.link(stereo.right)
46
47 spatialLocationCalculator.passthroughDepth.link(xoutDepth.input)
48 stereo.depth.link(spatialLocationCalculator.inputDepth)
49
50 topLeft = dai.Point2f(0.4, 0.4)
51 bottomRight = dai.Point2f(0.6, 0.6)
52
53 spatialLocationCalculator.setWaitForConfigInput(False)
54 config = dai.SpatialLocationCalculatorConfigData()
55 config.depthThresholds.lowerThreshold = 100
56 config.depthThresholds.upperThreshold = 10000
57 config.roi = dai.Rect(topLeft, bottomRight)
58 spatialLocationCalculator.initialConfig.addROI(config)
59 spatialLocationCalculator.out.link(xoutSpatialData.input)
60 xinSpatialCalcConfig.out.link(spatialLocationCalculator.inputConfig)
61
62 # Pipeline defined, now the device is assigned and pipeline is started
63 device = dai.Device(pipeline)
64 device.startPipeline()
65
66 # Output queue will be used to get the depth frames from the outputs defined above
67 depthQueue = device.getOutputQueue(name="depth", maxSize=4, blocking=False)
68 spatialCalcQueue = device.getOutputQueue(name="spatialData", maxSize=4,
↳blocking=False)
69 spatialCalcConfigInQueue = device.getInputQueue("spatialCalcConfig")
70
71 color = (255, 255, 255)
72
73 print("Use WASD keys to move ROI!")
74
75 while True:
76     inDepth = depthQueue.get() # blocking call, will wait until a new data has arrived
77     inDepthAvg = spatialCalcQueue.get() # blocking call, will wait until a new data
↳has arrived
78
79     depthFrame = inDepth.getFrame()
80     depthFrameColor = cv2.normalize(depthFrame, None, 255, 0, cv2.NORM_INF, cv2.CV_
↳8UC1)
81     depthFrameColor = cv2.equalizeHist(depthFrameColor)
82     depthFrameColor = cv2.applyColorMap(depthFrameColor, cv2.COLORMAP_HOT)
83
84     spatialData = inDepthAvg.getSpatialLocations()

```

(continues on next page)

(continued from previous page)

```

85     for depthData in spatialData:
86         roi = depthData.config.roi
87         roi = roi.denormalize(width=depthFrameColor.shape[1], height=depthFrameColor.
↪shape[0])
88         xmin = int(roi.topLeft().x)
89         ymin = int(roi.topLeft().y)
90         xmax = int(roi.bottomRight().x)
91         ymax = int(roi.bottomRight().y)
92
93         fontType = cv2.FONT_HERSHEY_TRIPLEX
94         cv2.rectangle(depthFrameColor, (xmin, ymin), (xmax, ymax), color, cv2.FONT_
↪HERSHEY_SCRIPT_SIMPLEX)
95         cv2.putText(depthFrameColor, f"X: {int(depthData.spatialCoordinates.x)} mm",
↪(xmin + 10, ymin + 20), fontType, 0.5, color)
96         cv2.putText(depthFrameColor, f"Y: {int(depthData.spatialCoordinates.y)} mm",
↪(xmin + 10, ymin + 35), fontType, 0.5, color)
97         cv2.putText(depthFrameColor, f"Z: {int(depthData.spatialCoordinates.z)} mm",
↪(xmin + 10, ymin + 50), fontType, 0.5, color)
98
99
100     cv2.imshow("depth", depthFrameColor)
101
102     newConfig = False
103     key = cv2.waitKey(1)
104     if key == ord('q'):
105         break
106     elif key == ord('w'):
107         if topLeft.y - stepSize >= 0:
108             topLeft.y -= stepSize
109             bottomRight.y -= stepSize
110             newConfig = True
111     elif key == ord('a'):
112         if topLeft.x - stepSize >= 0:
113             topLeft.x -= stepSize
114             bottomRight.x -= stepSize
115             newConfig = True
116     elif key == ord('s'):
117         if bottomRight.y + stepSize <= 1:
118             topLeft.y += stepSize
119             bottomRight.y += stepSize
120             newConfig = True
121     elif key == ord('d'):
122         if bottomRight.x + stepSize <= 1:
123             topLeft.x += stepSize
124             bottomRight.x += stepSize
125             newConfig = True
126
127     if newConfig:
128         config.roi = dai.Rect(topLeft, bottomRight)
129         cfg = dai.SpatialLocationCalculatorConfig()
130         cfg.addROI(config)
131         spatialCalcConfigInQueue.send(cfg)

```

We're always happy to help with code or other questions you might have.

## 3.33 28 - Camera video high resolution

This example shows how to use high resolution video at low latency. Compared to *01 - RGB Preview*, this demo outputs NV12 frames whereas preview frames are BGR and are not suited for larger resolution (eg. 2000x1000). Preview is more suitable for either NN or visualization purposes.

### 3.33.1 Setup

Please run the following command to install the required dependencies

```
python3 -m pip install -U pip
python3 -m pip install opencv-python
python3 -m pip install -U --force-reinstall depthai
```

For additional information, please follow *installation guide*

### 3.33.2 Source code

Also available on GitHub

```
1  #!/usr/bin/env python3
2
3  import cv2
4  import depthai as dai
5  import numpy as np
6
7  # Start defining a pipeline
8  pipeline = dai.Pipeline()
9
10 # Define a source - color camera
11 colorCam = pipeline.createColorCamera()
12 colorCam.setBoardSocket(dai.CameraBoardSocket.RGB)
13 colorCam.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
14 colorCam.setVideoSize(1920, 1080)
15
16 # Create output
17 xoutVideo = pipeline.createXLinkOut()
18 xoutVideo.setStreamName("video")
19 xoutVideo.input.setBlocking(False)
20 xoutVideo.input.setQueueSize(1)
21
22 colorCam.video.link(xoutVideo.input)
23
24 # Pipeline defined, now the device is connected to
25 with dai.Device(pipeline) as device:
26     # Start pipeline
27     device.startPipeline()
28     video = device.getOutputQueue(name="video", maxSize=1, blocking=False)
29
30     while True:
31         # Get preview and video frames
32         videoIn = video.get()
33
34         # Get BGR frame from NV12 encoded video frame to show with opencv
```

(continues on next page)

(continued from previous page)

```

35     # Visualizing the frame on slower hosts might have overhead
36     cv2.imshow("video", videoIn.getCvFrame())
37
38     if cv2.waitKey(1) == ord('q'):
39         break

```

We're always happy to help with code or other questions you might have.

## 3.34 Python API Reference

### Classes:

<i>ADatatype</i>	Abstract message
<i>Asset</i>	Asset is identified with string key and can store arbitrary binary data
<i>AssetManager</i>	AssetManager can store assets and serialize
<i>Buffer</i>	Base message - buffer of binary data
<i>CameraBoardSocket</i>	Which Camera socket to use.
<i>CameraControl</i>	CameraControl message Specifies various camera control commands like:
<i>CameraImageOrientation</i>	Camera sensor image orientation / pixel readout.
<i>ChipTemperature</i>	Chip temperature information.
<i>ColorCamera</i>	ColorCamera node.
<i>ColorCameraProperties</i>	Specify ColorCamera options such as camera ID, ...
<i>CpuUsage</i>	CpuUsage structure
<i>DataInputQueue</i>	Access to send messages through XLink stream
<i>DataOutputQueue</i>	Access to receive messages coming from XLink stream
<i>DetectionNetwork</i>	DetectionNetwork.
<i>DetectionNetworkProperties</i>	Properties for DetectionNetwork
<i>Device</i>	Represents the DepthAI device with the methods to interact with it.
<i>DeviceBootloader</i>	Represents the DepthAI bootloader with the methods to interact with it.
<i>DeviceDesc</i>	
<i>DeviceInfo</i>	
<i>GlobalProperties</i>	Specify properties which apply for whole pipeline
<i>ImageManip</i>	ImageManip node.
<i>ImageManipConfig</i>	ImageManipConfig message.
<i>ImgDetection</i>	
<i>ImgDetections</i>	ImgDetections message.
<i>ImgFrame</i>	ImgFrame message.
<i>LogLevel</i>	Members:
<i>MemoryInfo</i>	MemoryInfo structure
<i>MobileNetDetectionNetwork</i>	MobileNetDetectionNetwork node.
<i>MobileNetSpatialDetectionNetwork</i>	MobileNetSpatialDetectionNetwork.
<i>MonoCamera</i>	MonoCamera node.
<i>MonoCameraProperties</i>	Specify MonoCamera options such as camera ID, ...
<i>NNData</i>	NNData message.
<i>NeuralNetwork</i>	NeuralNetwork node.

continues on next page

Table 1 – continued from previous page

<i>NeuralNetworkProperties</i>	Specify NeuralNetwork options such as blob path, ...
<i>Node</i>	Abstract Node
<i>OpenVINO</i>	Support for basic OpenVINO related actions like version identification of neural network blobs,...
<i>Pipeline</i>	Represents the pipeline, set of nodes and connections between them
<i>Point2f</i>	Point2f structure
<i>Point3f</i>	Point3f structure
<i>RawBuffer</i>	
<i>RawCameraControl</i>	
<i>RawImageManipConfig</i>	
<i>RawImgDetections</i>	
<i>RawImgFrame</i>	
<i>RawNNData</i>	
<i>RawSpatialImgDetections</i>	
<i>RawSystemInformation</i>	System information of device
<i>Rect</i>	Rect structure
<i>RotatedRect</i>	
<i>SPIOut</i>	SPIOut node.
<i>Size2f</i>	
<i>SpatialDetectionNetwork</i>	SpatialDetectionNetwork node.
<i>SpatialDetectionNetworkProperties</i>	Properties for SpatialDetectionNetwork
<i>SpatialImgDetection</i>	Spatial image detection structure
<i>SpatialImgDetections</i>	SpatialImgDetections message.
<i>SpatialLocationCalculator</i>	SpatialLocationCalculator node.
<i>SpatialLocationCalculatorConfig</i>	SpatialLocationCalculatorConfig message.
<i>SpatialLocationCalculatorConfigData</i>	
<i>SpatialLocationCalculatorConfigThresholds</i>	Spatial location configuration thresholds structure
<i>SpatialLocationCalculatorData</i>	SpatialLocationCalculatorData message.
<i>SpatialLocationCalculatorProperties</i>	Specify SpatialLocationCalculator options
<i>SpatialLocations</i>	Spatial location information structure
<i>StereoDepth</i>	StereoDepth node.
<i>StereoDepthProperties</i>	Specify StereoDepth options
<i>SystemInformation</i>	SystemInformation message.
<i>SystemLogger</i>	SystemLogger node.
<i>SystemLoggerProperties</i>	
<i>TensorInfo</i>	
<i>Timestamp</i>	
<i>VideoEncoder</i>	VideoEncoder node.
<i>VideoEncoderProperties</i>	Specify VideoEncoder options such as profile, bitrate, ...
<i>XLinkConnection</i>	
<i>XLinkDeviceState</i>	Members:
<i>XLinkIn</i>	XLinkIn node.
<i>XLinkOut</i>	XLinkOut node.
<i>XLinkPlatform</i>	Members:
<i>XLinkProtocol</i>	Members:
<i>YoloDetectionNetwork</i>	YoloDetectionNetwork node.
<i>YoloSpatialDetectionNetwork</i>	YoloSpatialDetectionNetwork.

**class** depthai.**ADatatype**



Bases: `pybind11_builtins.pybind11_object`

Abstract message

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getRaw(self)</code>	

`__init__ (*args, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

`getRaw (self: depthai.ADatatype) → depthai.RawBuffer`

**class depthai.Asset**

Bases: `pybind11_builtins.pybind11_object`

Asset is identified with string key and can store arbitrary binary data

**Methods:**

<code>__init__(*args, **kwargs)</code>	Overloaded function.
--	----------------------

**Attributes:**

<code>alignment</code>
<code>data</code>
<code>key</code>

`__init__ (*args, **kwargs)`  
Overloaded function.

1. `__init__(self: depthai.Asset) -> None`
2. `__init__(self: depthai.Asset, arg0: str) -> None`

**property alignment**

**property data**

**property key**

**class depthai.AssetManager**

Bases: `pybind11_builtins.pybind11_object`

AssetManager can store assets and serialize

**Methods:**

<code>__init__(self)</code>	
<code>add(*args, **kwargs)</code>	Overloaded function.
<code>addExisting(self, assets)</code>	Adds all assets in an array to the AssetManager
<code>get(*args, **kwargs)</code>	Overloaded function.
<code>getAll(*args, **kwargs)</code>	Overloaded function.
<code>remove(self, key)</code>	Removes asset with key
<code>set(self, key, asset)</code>	Adds or overwrites existing asset with a specified key.

continues on next page

Table 5 – continued from previous page

---

<code>size(self)</code>	<b>returns</b> Number of asset stored in the AssetManager
-------------------------	---

---

`__init__ (self: depthai.AssetManager) → None`

**add** (\*args, \*\*kwargs)  
Overloaded function.

1. `add(self: depthai.AssetManager, asset: depthai.Asset) -> None`  
Adds an asset object to AssetManager.

**Parameter asset:** Asset to add

2. `add(self: depthai.AssetManager, key: str, asset: depthai.Asset) -> None`  
Adds an asset object to AssetManager with a specified key. Key value will be assigned to an Asset as well  
If asset with key already exists, the function throws an error

**Parameter key:** Key under which the asset should be stored

**Parameter asset:** Asset to store

**addExisting** (self: depthai.AssetManager, assets: List[depthai.Asset]) → None  
Adds all assets in an array to the AssetManager

**Parameter assets:** Vector of assets to add

**get** (\*args, \*\*kwargs)  
Overloaded function.

1. `get(self: depthai.AssetManager, key: str) -> depthai.Asset`  
**Returns** Asset assigned to the specified key or throws an error otherwise
2. `get(self: depthai.AssetManager, key: str) -> depthai.Asset`  
**Returns** Asset assigned to the specified key or throws an error otherwise

**getAll** (\*args, \*\*kwargs)  
Overloaded function.

1. `getAll(self: depthai.AssetManager) -> List[depthai.Asset]`  
**Returns** All asset stored in the AssetManager
2. `getAll(self: depthai.AssetManager) -> List[depthai.Asset]`  
**Returns** All asset stored in the AssetManager

**remove** (self: depthai.AssetManager, key: str) → None  
Removes asset with key

**Parameter key:** Key of asset to remove

---

**set** (*self*: depthai.AssetManager, *key*: *str*, *asset*: depthai.Asset) → None

Adds or overwrites existing asset with a specified key.

**Parameter key:** Key under which the asset should be stored

**Parameter asset:** Asset to store

**size** (*self*: depthai.AssetManager) → int

**Returns** Number of asset stored in the AssetManager

**class** depthai.Buffer

Bases: *depthai.ADatatype*

Base message - buffer of binary data

**Methods:**

<code>__init__(self)</code>	Creates Buffer message
<code>getData(self)</code>	<b>returns</b> Reference to internal buffer
<code>setData(*args, **kwargs)</code>	Overloaded function.

`__init__` (*self*: depthai.Buffer) → None

Creates Buffer message

**getData** (*self*: *object*) → numpy.ndarray[numpy.uint8]

**Returns** Reference to internal buffer

**setData** (*\*args*, *\*\*kwargs*)

Overloaded function.

1. setData(*self*: depthai.Buffer, *arg0*: List[int]) -> None

**Parameter data:** Copies data to internal buffer

2. setData(*self*: depthai.Buffer, *arg0*: numpy.ndarray[numpy.uint8]) -> None

**Parameter data:** Copies data to internal buffer

**class** depthai.CameraBoardSocket

Bases: *pybind11\_builtins.pybind11\_object*

Which Camera socket to use.

AUTO denotes that the decision will be made by device

Members:

AUTO

RGB

LEFT

RIGHT

**Attributes:**

---

*AUTO*

---

*LEFT*

---

*RGB*

---

*RIGHT*

---

*name*

---

*value*

---

**Methods:**

---

*\_\_init\_\_*(self, value)

---

**AUTO** = <CameraBoardSocket.AUTO: -1>**LEFT** = <CameraBoardSocket.LEFT: 1>**RGB** = <CameraBoardSocket.RGB: 0>**RIGHT** = <CameraBoardSocket.RIGHT: 2>*\_\_init\_\_* (self: depthai.CameraBoardSocket, value: int) → None**property name****property value****class** depthai.CameraControlBases: *depthai.Buffer*

CameraControl message Specifies various camera control commands like:

- Still capture
- Auto focus
- Anti banding
- Auto white balance
- Scene
- Effect
- ...

**Classes:**

---

*AntiBandingMode*

---

Members:

---

*AutoFocusMode*

---

Members:

---

*AutoWhiteBalanceMode*

---

Members:

---

*EffectMode*

---

Members:

---

*SceneMode*

---

Members:

**Methods:**

---

*\_\_init\_\_*(self)

---

Construct CameraControl message

---

*getCaptureStill*(self)

---

Check whether command to capture a still is set

---

*setAntiBandingMode*(self, mode)

---

Set a command to specify auto banding mode

continues on next page

Table 10 – continued from previous page

<i>setAutoExposureCompensation</i> (self, compensation)	Set a command to specify auto exposure compensation
<i>setAutoExposureEnable</i> (self)	Set a command to enable auto exposure
<i>setAutoExposureLock</i> (self, lock)	Set a command to specify lock auto exposure
<i>setAutoExposureRegion</i> (self, startX, startY, ...)	Set a command to specify auto exposure region in pixels
<i>setAutoFocusMode</i> (self, mode)	Set a command to specify autofocus mode
<i>setAutoFocusRegion</i> (self, startX, startY, ...)	Set a command to specify focus region in pixels
<i>setAutoFocusTrigger</i> (self)	Set a command to trigger autofocus
<i>setAutoWhiteBalanceLock</i> (self, lock)	Set a command to specify auto white balance lock
<i>setAutoWhiteBalanceMode</i> (self, mode)	Set a command to specify auto white balance mode
<i>setBrightness</i> (self, value)	Set a command to specify auto white balance lock
<i>setCaptureStill</i> (self, capture)	Set a command to capture a still image
<i>setChromaDenoise</i> (self, value)	Set a command to specify chroma denoise value
<i>setContrast</i> (self, value)	Set a command to specify auto white balance lock
<i>setEffectMode</i> (self, mode)	Set a command to specify effect mode
<i>setLumaDenoise</i> (self, value)	Set a command to specify luma denoise value
<i>setManualExposure</i> (self, exposureTimeUs, ...)	Set a command to manually specify exposure
<i>setManualFocus</i> (self, lensPosition)	Set a command to specify manual focus position
<i>setNoiseReductionStrength</i> (self, value)	Set a command to specify noise reduction strength
<i>setSaturation</i> (self, value)	Set a command to specify saturation value
<i>setSceneMode</i> (self, mode)	Set a command to specify scene mode
<i>setSharpness</i> (self, value)	Set a command to specify sharpness value
<i>setStartStreaming</i> (self)	Set a command to start streaming
<i>setStopStreaming</i> (self)	Set a command to stop streaming

**class AntiBandingMode**Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

MAINS\_50\_HZ

MAINS\_60\_HZ

AUTO

Attributes:

<i>AUTO</i>
<i>MAINS_50_HZ</i>
<i>MAINS_60_HZ</i>
<i>OFF</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__</i> (self, value)
-------------------------------

**AUTO = <AntiBandingMode.AUTO: 3>**

**MAINS\_50\_HZ** = <AntiBandingMode.MAINS\_50\_HZ: 1>

**MAINS\_60\_HZ** = <AntiBandingMode.MAINS\_60\_HZ: 2>

**OFF** = <AntiBandingMode.OFF: 0>

`__init__(self: depthai.RawCameraControl.AntiBandingMode, value: int) → None`

property name

property value

**class AutoFocusMode**

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

AUTO

MACRO

CONTINUOUS\_VIDEO

CONTINUOUS\_PICTURE

EDOF

**Attributes:**

---

*AUTO*

---

*CONTINUOUS\_PICTURE*

---

*CONTINUOUS\_VIDEO*

---

*EDOF*

---

*MACRO*

---

*OFF*

---

*name*

---

*value*

---

**Methods:**

---

`__init__(self, value)`

---

**AUTO** = <AutoFocusMode.AUTO: 1>

**CONTINUOUS\_PICTURE** = <AutoFocusMode.CONTINUOUS\_PICTURE: 4>

**CONTINUOUS\_VIDEO** = <AutoFocusMode.CONTINUOUS\_VIDEO: 3>

**EDOF** = <AutoFocusMode.EDOF: 5>

**MACRO** = <AutoFocusMode.MACRO: 2>

**OFF** = <AutoFocusMode.OFF: 0>

`__init__(self: depthai.RawCameraControl.AutoFocusMode, value: int) → None`

property name

property value

**class AutoWhiteBalanceMode**

Bases: `pybind11_builtins.pybind11_object`

## Members:

OFF  
 AUTO  
 INCANDESCENT  
 FLUORESCENT  
 WARM\_FLUORESCENT  
 DAYLIGHT  
 CLOUDY\_DAYLIGHT  
 TWILIGHT  
 SHADE

## Attributes:

<i>AUTO</i>
<i>CLOUDY_DAYLIGHT</i>
<i>DAYLIGHT</i>
<i>FLUORESCENT</i>
<i>INCANDESCENT</i>
<i>OFF</i>
<i>SHADE</i>
<i>TWILIGHT</i>
<i>WARM_FLUORESCENT</i>
<i>name</i>
<i>value</i>

## Methods:

<code>__init__(self, value)</code>
------------------------------------

```

AUTO = <AutoWhiteBalanceMode.AUTO: 1>
CLOUDY_DAYLIGHT = <AutoWhiteBalanceMode.CLOUDY_DAYLIGHT: 6>
DAYLIGHT = <AutoWhiteBalanceMode.DAYLIGHT: 5>
FLUORESCENT = <AutoWhiteBalanceMode.FLUORESCENT: 3>
INCANDESCENT = <AutoWhiteBalanceMode.INCANDESCENT: 2>
OFF = <AutoWhiteBalanceMode.OFF: 0>
SHADE = <AutoWhiteBalanceMode.SHADE: 8>
TWILIGHT = <AutoWhiteBalanceMode.TWILIGHT: 7>
WARM_FLUORESCENT = <AutoWhiteBalanceMode.WARM_FLUORESCENT: 4>
__init__(self: depthai.RawCameraControl.AutoWhiteBalanceMode, value: int) → None
property name
property value

```

**class EffectMode**Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

MONO

NEGATIVE

SOLARIZE

SEPIA

POSTERIZE

WHITEBOARD

BLACKBOARD

AQUA

**Attributes:**

<i>AQUA</i>
<i>BLACKBOARD</i>
<i>MONO</i>
<i>NEGATIVE</i>
<i>OFF</i>
<i>POSTERIZE</i>
<i>SEPIA</i>
<i>SOLARIZE</i>
<i>WHITEBOARD</i>
<i>name</i>
<i>value</i>

**Methods:**

<i>__init__</i> (self, value)
-------------------------------

**AQUA** = <EffectMode.AQUA: 8>**BLACKBOARD** = <EffectMode.BLACKBOARD: 7>**MONO** = <EffectMode.MONO: 1>**NEGATIVE** = <EffectMode.NEGATIVE: 2>**OFF** = <EffectMode.OFF: 0>**POSTERIZE** = <EffectMode.POSTERIZE: 5>**SEPIA** = <EffectMode.SEPIA: 4>**SOLARIZE** = <EffectMode.SOLARIZE: 3>**WHITEBOARD** = <EffectMode.WHITEBOARD: 6>*\_\_init\_\_*(self: `depthai.RawCameraControl.EffectMode`, value: *int*) → None

property name



**property value**

**class SceneMode**

Bases: `pybind11_builtins.pybind11_object`

Members:

UNSUPPORTED  
FACE\_PRIORITY  
ACTION  
PORTRAIT  
LANDSCAPE  
NIGHT  
NIGHT\_PORTRAIT  
THEATRE  
BEACH  
SNOW  
SUNSET  
STEADYPHOTO  
FIREWORKS  
SPORTS  
PARTY  
CANDLELIGHT  
BARCODE

**Attributes:**

<i>ACTION</i>
<i>BARCODE</i>
<i>BEACH</i>
<i>CANDLELIGHT</i>
<i>FACE_PRIORITY</i>
<i>FIREWORKS</i>
<i>LANDSCAPE</i>
<i>NIGHT</i>
<i>NIGHT_PORTRAIT</i>
<i>PARTY</i>
<i>PORTRAIT</i>
<i>SNOW</i>
<i>SPORTS</i>
<i>STEADYPHOTO</i>
<i>SUNSET</i>
<i>THEATRE</i>
<i>UNSUPPORTED</i>
<i>name</i>
<i>value</i>

**Methods:**


---

<code>__init__(self, value)</code>	
------------------------------------	--

---

<code>ACTION = &lt;SceneMode.ACTION: 2&gt;</code>	
<code>BARCODE = &lt;SceneMode.BARCODE: 16&gt;</code>	
<code>BEACH = &lt;SceneMode.BEACH: 8&gt;</code>	
<code>CANDLELIGHT = &lt;SceneMode.CANDLELIGHT: 15&gt;</code>	
<code>FACE_PRIORITY = &lt;SceneMode.FACE_PRIORITY: 1&gt;</code>	
<code>FIREWORKS = &lt;SceneMode.FIREWORKS: 12&gt;</code>	
<code>LANDSCAPE = &lt;SceneMode.LANDSCAPE: 4&gt;</code>	
<code>NIGHT = &lt;SceneMode.NIGHT: 5&gt;</code>	
<code>NIGHT_PORTRAIT = &lt;SceneMode.NIGHT_PORTRAIT: 6&gt;</code>	
<code>PARTY = &lt;SceneMode.PARTY: 14&gt;</code>	
<code>PORTRAIT = &lt;SceneMode.PORTRAIT: 3&gt;</code>	
<code>SNOW = &lt;SceneMode.SNOW: 9&gt;</code>	
<code>SPORTS = &lt;SceneMode.SPORTS: 13&gt;</code>	
<code>STEADYPHOTO = &lt;SceneMode.STEADYPHOTO: 11&gt;</code>	
<code>SUNSET = &lt;SceneMode.SUNSET: 10&gt;</code>	
<code>THEATRE = &lt;SceneMode.THEATRE: 7&gt;</code>	
<code>UNSUPPORTED = &lt;SceneMode.UNSUPPORTED: 0&gt;</code>	
<code>__init__(self: depthai.RawCameraControl.SceneMode, value: int) → None</code>	
<b>property name</b>	
<b>property value</b>	
<code>__init__(self: depthai.CameraControl) → None</code>	Construct CameraControl message
<code>getCaptureStill(self: depthai.CameraControl) → bool</code>	Check whether command to capture a still is set
<b>Returns</b>	True if capture still command is set
<code>setAntiBandingMode(self: depthai.CameraControl, mode: depthai.RawCameraControl.AntiBandingMode) → None</code>	Set a command to specify auto banding mode
<b>Parameter mode:</b>	Auto banding mode to use
<code>setAutoExposureCompensation(self: depthai.CameraControl, compensation: int) → None</code>	Set a command to specify auto exposure compenstaion
<b>Parameter compensation:</b>	Compensation value between -128..127
<code>setAutoExposureEnable(self: depthai.CameraControl) → None</code>	Set a command to enable auto exposure
<code>setAutoExposureLock(self: depthai.CameraControl, lock: bool) → None</code>	Set a command to specify lock auto exposure

**Parameter lock:** Auto exposure lock mode enabled or disabled

**setAutoExposureRegion** (*self*: [depthai.CameraControl](#), *startX*: *int*, *startY*: *int*, *width*: *int*, *height*: *int*) → [None](#)

Set a command to specify auto exposure region in pixels

**Parameter startX:** X coordinate of top left corner of region

**Parameter startY:** Y coordinate of top left corner of region

**Parameter width:** Region width

**Parameter height:** Region height

**setAutoFocusMode** (*self*: [depthai.CameraControl](#), *mode*: [depthai.RawCameraControl.AutoFocusMode](#)) → [None](#)

Set a command to specify autofocus mode

**setAutoFocusRegion** (*self*: [depthai.CameraControl](#), *startX*: *int*, *startY*: *int*, *width*: *int*, *height*: *int*) → [None](#)

Set a command to specify focus region in pixels

**Parameter startX:** X coordinate of top left corner of region

**Parameter startY:** Y coordinate of top left corner of region

**Parameter width:** Region width

**Parameter height:** Region height

**setAutoFocusTrigger** (*self*: [depthai.CameraControl](#)) → [None](#)

Set a command to trigger autofocus

**setAutoWhiteBalanceLock** (*self*: [depthai.CameraControl](#), *lock*: *bool*) → [None](#)

Set a command to specify auto white balance lock

**Parameter lock:** Auto white balance lock mode enabled or disabled

**setAutoWhiteBalanceMode** (*self*: [depthai.CameraControl](#), *mode*: [depthai.RawCameraControl.AutoWhiteBalanceMode](#)) → [None](#)

Set a command to specify auto white balance mode

**Parameter mode:** Auto white balance mode to use

**setBrightness** (*self*: [depthai.CameraControl](#), *value*: *int*) → [None](#)

Set a command to specify auto white balance lock

**Parameter lock:** Auto white balance lock mode enabled or disabled

**setCaptureStill** (*self*: [depthai.CameraControl](#), *capture*: *bool*) → [None](#)

Set a command to capture a still image

**setChromaDenoise** (*self*: [depthai.CameraControl](#), *value*: *int*) → [None](#)

Set a command to specify chroma denoise value

**Parameter value:** Chroma denoise

**setContrast** (*self*: [depthai.CameraControl](#), *value*: *int*) → [None](#)

Set a command to specify auto white balance lock

**Parameter lock:** Auto white balance lock mode enabled or disabled

**setEffectMode** (*self*: [depthai.CameraControl](#), *mode*: [depthai.RawCameraControl.EffectMode](#)) → [None](#)

Set a command to specify effect mode

**Parameter mode:** Effect mode

**setLumaDenoise** (*self*: `depthai.CameraControl`, *value*: `int`) → `None`

Set a command to specify luma denoise value

**Parameter value:** Luma denoise

**setManualExposure** (*self*: `depthai.CameraControl`, *exposureTimeUs*: `int`, *sensitivityIso*: `int`) → `None`

Set a command to manually specify exposure

**Parameter exposureTimeUs:** Exposure time in microseconds

**Parameter sensitivityIso:** Sensitivity as ISO value

**setManualFocus** (*self*: `depthai.CameraControl`, *lensPosition*: `int`) → `None`

Set a command to specify manual focus position

**Parameter lensPosition:** specify lens position 0..255

**setNoiseReductionStrength** (*self*: `depthai.CameraControl`, *value*: `int`) → `None`

Set a command to specify noise reduction strength

**Parameter value:** Noise reduction strength

**setSaturation** (*self*: `depthai.CameraControl`, *value*: `int`) → `None`

Set a command to specify saturation value

**Parameter value:** Saturation

**setSceneMode** (*self*: `depthai.CameraControl`, *mode*: `depthai.RawCameraControl.SceneMode`) → `None`

Set a command to specify scene mode

**Parameter mode:** Scene mode

**setSharpness** (*self*: `depthai.CameraControl`, *value*: `int`) → `None`

Set a command to specify sharpness value

**Parameter value:** Sharpness

**setStartStreaming** (*self*: `depthai.CameraControl`) → `None`

Set a command to start streaming

**setStopStreaming** (*self*: `depthai.CameraControl`) → `None`

Set a command to stop streaming

**class** `depthai.CameraImageOrientation`

Bases: `pybind11_builtins.pybind11_object`

Camera sensor image orientation / pixel readout. This exposes direct sensor settings. 90 or 270 degrees rotation is not available.

AUTO denotes that the decision will be made by device (e.g. on OAK-1/megaAI: ROTATE\_180\_DEG).

Members:

AUTO

NORMAL

HORIZONTAL\_MIRROR

VERTICAL\_FLIP

ROTATE\_180\_DEG

Attributes:

---

*AUTO*

---

*HORIZONTAL\_MIRROR*

---

*NORMAL*

---

*ROTATE\_180\_DEG*

---

*VERTICAL\_FLIP*

---

*name*

---

*value*

---

**Methods:**


---

`__init__(self, value)`

---

```
AUTO = <CameraImageOrientation.AUTO: -1>
HORIZONTAL_MIRROR = <CameraImageOrientation.HORIZONTAL_MIRROR: 1>
NORMAL = <CameraImageOrientation.NORMAL: 0>
ROTATE_180_DEG = <CameraImageOrientation.ROTATE_180_DEG: 3>
VERTICAL_FLIP = <CameraImageOrientation.VERTICAL_FLIP: 2>
__init__(self: depthai.CameraImageOrientation, value: int) → None
property name
property value
```

**class** depthai.ChipTemperature

Bases: pybind11\_builtins.pybind11\_object

Chip temperature information.

Multiple temperature measurement points and their average

**Methods:**


---

`__init__(self)`

---

**Attributes:**


---

*average*

---

*css*

---

*dss*

---

*mss*

---

*upa*

---

```
__init__(self: depthai.ChipTemperature) → None
property average
property css
property dss
property mss
property upa
```

**class** `depthai.ColorCamera`Bases: `depthai.Node`

ColorCamera node. For use with color sensors.

**Classes:**

<i>Properties</i>	alias of <code>depthai.ColorCameraProperties</code>
-------------------	---

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBoardSocket(self)</code>	Retrieves which board socket to use
<code>getCamId(self)</code>	
<code>getColorOrder(self)</code>	Get color order of preview output frames.
<code>getFp16(self)</code>	Get fp16 (0..255) data of preview output frames
<code>getFps(self)</code>	Get rate at which camera should produce frames
<code>getImageOrientation(self)</code>	Get camera image orientation
<code>getInterleaved(self)</code>	Get planar or interleaved data of preview output frames
<code>getPreviewHeight(self)</code>	Get preview height
<code>getPreviewKeepAspectRatio(self)</code>	<b>See also:</b> <code>setPreviewKeepAspectRatio</code>
<code>getPreviewSize(self)</code>	Get preview size as tuple
<code>getPreviewWidth(self)</code>	Get preview width
<code>getResolution(self)</code>	Get sensor resolution
<code>getResolutionHeight(self)</code>	Get sensor resolution height
<code>getResolutionSize(self)</code>	Get sensor resolution as size
<code>getResolutionWidth(self)</code>	Get sensor resolution width
<code>getSensorCrop(self)</code>	<b>returns</b> Sensor top left crop coordinates
<code>getSensorCropX(self)</code>	Get sensor top left x crop coordinate
<code>getSensorCropY(self)</code>	Get sensor top left y crop coordinate
<code>getStillHeight(self)</code>	Get still height
<code>getStillSize(self)</code>	Get still size as tuple
<code>getStillWidth(self)</code>	Get still width
<code>getVideoHeight(self)</code>	Get video height
<code>getVideoSize(self)</code>	Get video size as tuple
<code>getVideoWidth(self)</code>	Get video width
<code>getWaitForConfigInput(self)</code>	<b>See also:</b> <code>setWaitForConfigInput</code>
<code>sensorCenterCrop(self)</code>	Specify sensor center crop.
<code>setBoardSocket(self, boardSocket)</code>	Specify which board socket to use
<code>setCamId(self, arg0)</code>	
<code>setColorOrder(self, colorOrder)</code>	Set color order of preview output images.
<code>setFp16(self, fp16)</code>	Set fp16 (0..255) data type of preview output frames
<code>setFps(self, fps)</code>	Set rate at which camera should produce frames
<code>setImageOrientation(self, boardSocket)</code>	Set camera image orientation

continues on next page

Table 26 – continued from previous page

<i>setInterleaved</i> (self, interleaved)	Set planar or interleaved data of preview output frames
<i>setPreviewKeepAspectRatio</i> (self, keep)	Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.
<i>setPreviewSize</i> (self, width, height)	Set preview output size
<i>setResolution</i> (self, resolution)	Set sensor resolution
<i>setSensorCrop</i> (self, x, y)	Specifies sensor crop rectangle
<i>setStillSize</i> (self, width, height)	Set still output size
<i>setVideoSize</i> (self, width, height)	Set video output size
<i>setWaitForConfigInput</i> (self, wait)	Specify to wait until inputConfig receives a configuration message, before sending out a frame.

**Attributes:**

<i>initialControl</i>	Initial control options to apply to sensor
<i>inputConfig</i>	Input for ImageManipConfig message, which can modify crop parameters in runtime
<i>inputControl</i>	Input for CameraControl message, which can modify camera parameters in runtime
<i>preview</i>	Outputs ImgFrame message that carries BGR/RGB planar/interleaved encoded frame data.
<i>still</i>	Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.
<i>video</i>	Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

**Properties**alias of *depthai.ColorCameraProperties* **Classes:**

ColorOrder	For 24 bit color these can be either RGB or BGR
SensorResolution	Select the camera sensor resolution

**Methods:**

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

**Attributes:**

<i>boardSocket</i>
<i>colorOrder</i>
<i>fps</i>
<i>initialControl</i>
<i>interleaved</i>
<i>previewHeight</i>
<i>previewWidth</i>
<i>resolution</i>
<i>sensorCropX</i>
<i>sensorCropY</i>
<i>stillHeight</i>

continues on next page

Table 30 – continued from previous page

stillWidth
videoHeight
videoWidth

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**getBoardSocket** (self: depthai.ColorCamera) → dai::CameraBoardSocket

Retrieves which board socket to use

**Returns** Board socket to use

**getCamId** (self: depthai.ColorCamera) → int

**getColorOrder** (self: depthai.ColorCamera) → dai::ColorCameraProperties::ColorOrder

Get color order of preview output frames. RGB or BGR

**getFp16** (self: depthai.ColorCamera) → bool

Get fp16 (0..255) data of preview output frames

**getFps** (self: depthai.ColorCamera) → float

Get rate at which camera should produce frames

**Returns** Rate in frames per second

**getImageOrientation** (self: depthai.ColorCamera) → dai::CameraImageOrientation

Get camera image orientation

**getInterleaved** (self: depthai.ColorCamera) → bool

Get planar or interleaved data of preview output frames

**getPreviewHeight** (self: depthai.ColorCamera) → int

Get preview height

**getPreviewKeepAspectRatio** (self: depthai.ColorCamera) → bool

**See also:**

setPreviewKeepAspectRatio

**Returns** Preview keep aspect ratio option

**getPreviewSize** (self: depthai.ColorCamera) → Tuple[int, int]

Get preview size as tuple

**getPreviewWidth** (self: depthai.ColorCamera) → int

Get preview width

**getResolution** (self: depthai.ColorCamera) → dai::ColorCameraProperties::SensorResolution

Get sensor resolution

**getResolutionHeight** (self: depthai.ColorCamera) → int

Get sensor resolution height

**getResolutionSize** (self: depthai.ColorCamera) → Tuple[int, int]

Get sensor resolution as size

**getResolutionWidth** (self: depthai.ColorCamera) → int

Get sensor resolution width

**getSensorCrop** (self: depthai.ColorCamera) → Tuple[float, float]



**Returns** Sensor top left crop coordinates

**getSensorCropX** (*self*: [depthai.ColorCamera](#)) → float  
Get sensor top left x crop coordinate

**getSensorCropY** (*self*: [depthai.ColorCamera](#)) → float  
Get sensor top left y crop coordinate

**getStillHeight** (*self*: [depthai.ColorCamera](#)) → int  
Get still height

**getStillSize** (*self*: [depthai.ColorCamera](#)) → Tuple[int, int]  
Get still size as tuple

**getStillWidth** (*self*: [depthai.ColorCamera](#)) → int  
Get still width

**getVideoHeight** (*self*: [depthai.ColorCamera](#)) → int  
Get video height

**getVideoSize** (*self*: [depthai.ColorCamera](#)) → Tuple[int, int]  
Get video size as tuple

**getVideoWidth** (*self*: [depthai.ColorCamera](#)) → int  
Get video width

**getWaitForConfigInput** (*self*: [depthai.ColorCamera](#)) → bool

**See also:**

[setWaitForConfigInput](#)

**Returns** True if wait for inputConfig message, false otherwise

**property initialControl**  
Initial control options to apply to sensor

**property inputConfig**  
Input for ImageManipConfig message, which can modify crop paremeters in runtime  
Default queue is non-blocking with size 8

**property inputControl**  
Input for CameraControl message, which can modify camera parameters in runtime  
Default queue is blocking with size 8

**property preview**  
Outputs ImgFrame message that carries BGR/RGB planar/interleaved encoded frame data.  
Suitable for use with NeuralNetwork node

**sensorCenterCrop** (*self*: [depthai.ColorCamera](#)) → None  
Specify sensor center crop. Resolution size / video size

**setBoardSocket** (*self*: [depthai.ColorCamera](#), *boardSocket*: [dai::CameraBoardSocket](#)) → None  
Specify which board socket to use

**Parameter boardSocket:** Board socket to use

**setCamId** (*self*: [depthai.ColorCamera](#), *arg0*: int) → None

**setColorOrder** (*self*: *depthai.ColorCamera*, *colorOrder*: *dai::ColorCameraProperties::ColorOrder*)

→ *None*  
Set color order of preview output images. RGB or BGR

**setFp16** (*self*: *depthai.ColorCamera*, *fp16*: *bool*) → *None*

Set fp16 (0..255) data type of preview output frames

**setFps** (*self*: *depthai.ColorCamera*, *fps*: *float*) → *None*

Set rate at which camera should produce frames

**Parameter fps:** Rate in frames per second

**setImageOrientation** (*self*: *depthai.ColorCamera*, *boardSocket*: *dai::CameraImageOrientation*)

→ *None*  
Set camera image orientation

**setInterleaved** (*self*: *depthai.ColorCamera*, *interleaved*: *bool*) → *None*

Set planar or interleaved data of preview output frames

**setPreviewKeepAspectRatio** (*self*: *depthai.ColorCamera*, *keep*: *bool*) → *None*

Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.

**Parameter keep:** If true, a larger crop region will be considered to still be able to create the final image in the specified aspect ratio. Otherwise video size is resized to fit preview size

**setPreviewSize** (*self*: *depthai.ColorCamera*, *width*: *int*, *height*: *int*) → *None*

Set preview output size

**setResolution** (*self*: *depthai.ColorCamera*, *resolution*: *dai::ColorCameraProperties::SensorResolution*)

→ *None*  
Set sensor resolution

**setSensorCrop** (*self*: *depthai.ColorCamera*, *x*: *float*, *y*: *float*) → *None*

Specifies sensor crop rectangle

**Parameter x:** Top left X coordinate

**Parameter y:** Top left Y coordinate

**setStillSize** (*self*: *depthai.ColorCamera*, *width*: *int*, *height*: *int*) → *None*

Set still output size

**setVideoSize** (*self*: *depthai.ColorCamera*, *width*: *int*, *height*: *int*) → *None*

Set video output size

**setWaitForConfigInput** (*self*: *depthai.ColorCamera*, *wait*: *bool*) → *None*

Specify to wait until inputConfig receives a configuration message, before sending out a frame.

**Parameter wait:** True to wait for inputConfig message, false otherwise

**property still**

Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

The message is sent only when a CameraControl message arrives to inputControl with captureStill command set.

**property video**

Outputs ImgFrame message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Suitable for use with VideoEncoder node

**class** *depthai.ColorCameraProperties*

Bases: *pybind11\_builtins.pybind11\_object*

Specify ColorCamera options such as camera ID, ...

**Classes:**

<i>ColorOrder</i>	For 24 bit color these can be either RGB or BGR
<i>SensorResolution</i>	Select the camera sensor resolution

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<i>boardSocket</i>
<i>colorOrder</i>
<i>fps</i>
<i>initialControl</i>
<i>interleaved</i>
<i>previewHeight</i>
<i>previewWidth</i>
<i>resolution</i>
<i>sensorCropX</i>
<i>sensorCropY</i>
<i>stillHeight</i>
<i>stillWidth</i>
<i>videoHeight</i>
<i>videoWidth</i>

**class ColorOrder**Bases: `pybind11_builtins.pybind11_object`

For 24 bit color these can be either RGB or BGR

Members:

BGR

RGB

**Attributes:**

<i>BGR</i>
<i>RGB</i>
<i>name</i>
<i>value</i>

**Methods:**

<code>__init__(self, value)</code>
------------------------------------

**BGR** = `<ColorOrder.BGR: 0>`**RGB** = `<ColorOrder.RGB: 1>``__init__(self: depthai.ColorCameraProperties.ColorOrder, value: int) → None`**property name**

property value

**class** SensorResolution

Bases: `pybind11_builtins.pybind11_object`

Select the camera sensor resolution

Members:

THE\_1080\_P

THE\_4\_K

THE\_12\_MP

**Attributes:**

---

*THE\_1080\_P*

---

*THE\_12\_MP*

---

*THE\_4\_K*

---

*name*

---

*value*

---

**Methods:**

---

*\_\_init\_\_*(self, value)

---

THE\_1080\_P = <SensorResolution.THE\_1080\_P: 0>

THE\_12\_MP = <SensorResolution.THE\_12\_MP: 2>

THE\_4\_K = <SensorResolution.THE\_4\_K: 1>

*\_\_init\_\_*(self: `depthai.ColorCameraProperties.SensorResolution`, value: *int*) → None

property name

property value

*\_\_init\_\_*(\*args, \*\*kwargs)

Initialize self. See `help(type(self))` for accurate signature.

property boardSocket

property colorOrder

property fps

property initialControl

property interleaved

property previewHeight

property previewWidth

property resolution

property sensorCropX

property sensorCropY

property stillHeight

property stillWidth

**property** videoHeight

**property** videoWidth

**class** depthai.CpuUsage

Bases: pybind11\_builtins.pybind11\_object

CpuUsage structure

Average usage in percent and time span of the average (since last query)

**Methods:**

---

`__init__(self)`

---

**Attributes:**

---

`average`

---

`msTime`

---

`__init__(self: depthai.CpuUsage) → None`

**property** average

**property** msTime

**class** depthai.DataInputQueue

Bases: pybind11\_builtins.pybind11\_object

Access to send messages through XLink stream

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBlocking(self)</code>	Gets current queue behavior when full (maxSize)
<code>getMaxSize(self, arg0)</code>	Gets queue maximum size
<code>getName(self)</code>	Gets queues name
<code>send(*args, **kwargs)</code>	Overloaded function.
<code>setBlocking(self, blocking)</code>	Sets queue behavior when full (maxSize)
<code>setMaxSize(self, maxSize)</code>	Sets queue maximum size

`__init__ (*args, **kwargs)`  
Initialize self. See help(type(self)) for accurate signature.

**getBlocking** (*self*: depthai.DataInputQueue) → bool  
Gets current queue behavior when full (maxSize)

**Returns** true if blocking, false otherwise

**getMaxSize** (*self*: depthai.DataInputQueue, *arg0*: int) → int  
Gets queue maximum size

**Returns** Maximum queue size

**getName** (*self*: depthai.DataInputQueue) → str  
Gets queues name

**Returns** Queue name

**send** (*\*args, \*\*kwargs*)  
Overloaded function.

1. `send(self: depthai.DataInputQueue, msg: depthai.ADatatype) -> None`

Adds a message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

**Parameter msg:** Message to add to the queue

2. `send(self: depthai.DataInputQueue, rawMsg: depthai.RawBuffer) -> None`

Adds a raw message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

**Parameter rawMsg:** Message to add to the queue

**setBlocking** (*self: depthai.DataInputQueue, blocking: bool*) → None

Sets queue behavior when full (maxSize)

**Parameter blocking:** Specifies if block or overwrite the oldest message in the queue

**setMaxSize** (*self: depthai.DataInputQueue, maxSize: int*) → None

Sets queue maximum size

**Parameter maxSize:** Specifies maximum number of messages in the queue

**class** `depthai.DataOutputQueue`

Bases: `pybind11_builtins.pybind11_object`

Access to receive messages coming from XLink stream

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>addCallback(*args, **kwargs)</code>	Overloaded function.
<code>get(self)</code>	Block until a message is available.
<code>getAll(self)</code>	Block until at least one message in the queue.
<code>getBlocking(self)</code>	Gets current queue behavior when full (maxSize)
<code>getMaxSize(self, arg0)</code>	Gets queue maximum size
<code>getName(self)</code>	Gets queues name
<code>has(self)</code>	Check whether front of the queue has a message (isn't empty)
<code>removeCallback(self, callbackId)</code>	Removes a callback
<code>setBlocking(self, blocking)</code>	Sets queue behavior when full (maxSize)
<code>setMaxSize(self, maxSize)</code>	Sets queue maximum size
<code>tryGet(self)</code>	Try to retrieve message from queue.
<code>tryGetAll(self)</code>	Try to retrieve all messages in the queue.

`__init__ (*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

**addCallback** (*\*args, \*\*kwargs*)

Overloaded function.

1. `addCallback(self: depthai.DataOutputQueue, callback: std::function<void (std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, std::shared_ptr<dai::ADatatype>>)> -> int`

Adds a callback on message received

**Parameter callback:** Callback function with queue name and message pointer

**Returns** Callback id

2. `addCallback(self: depthai.DataOutputQueue, callback: std::function<void (std::shared_ptr<dai::ADatatype>>)> -> int`

Adds a callback on message received

**Parameter callback:** Callback function with message pointer

**Returns** Callback id

3. `addCallback(self: depthai.DataOutputQueue, callback: std::function<void ()> -> int`

Adds a callback on message received

**Parameter callback:** Callback function without any parameters

**Returns** Callback id

`get (self: depthai.DataOutputQueue) -> depthai.ADatatype`  
Block until a message is available.

**Returns** Message or nullptr if no message available

`getAll (self: depthai.DataOutputQueue) -> List[depthai.ADatatype]`  
Block until at least one message in the queue. Then return all messages from the queue.

**Returns** Vector of messages

`getBlocking (self: depthai.DataOutputQueue) -> bool`  
Gets current queue behavior when full (maxSize)

**Returns** true if blocking, false otherwise

`getMaxSize (self: depthai.DataOutputQueue, arg0: int) -> int`  
Gets queue maximum size

**Returns** Maximum queue size

`getName (self: depthai.DataOutputQueue) -> str`  
Gets queues name

**Returns** Queue name

`has (self: depthai.DataOutputQueue) -> bool`  
Check whether front of the queue has a message (isn't empty)

**Returns** true if queue isn't empty, false otherwise

`removeCallback (self: depthai.DataOutputQueue, callbackId: int) -> bool`  
Removes a callback

**Parameter callbackId:** Id of callback to be removed

**Returns** true if callback was removed, false otherwise

`setBlocking (self: depthai.DataOutputQueue, blocking: bool) -> None`  
Sets queue behavior when full (maxSize)

**Parameter blocking:** Specifies if block or overwrite the oldest message in the queue

**setMaxSize** (*self*: [depthai.DataOutputQueue](#), *maxSize*: *int*) → *None*

Sets queue maximum size

**Parameter maxSize:** Specifies maximum number of messages in the queue

**tryGet** (*self*: [depthai.DataOutputQueue](#)) → [depthai.ADatatype](#)

Try to retrieve message from queue. If no message available, return immediately with nullptr

**Returns** Message or nullptr if no message available

**tryGetAll** (*self*: [depthai.DataOutputQueue](#)) → List[[depthai.ADatatype](#)]

Try to retrieve all messages in the queue.

**Returns** Vector of messages

**class** [depthai.DetectionNetwork](#)

Bases: [depthai.NeuralNetwork](#)

DetectionNetwork. Base for different network specializations

**Classes:**

<i>Properties</i>	alias of <a href="#">depthai.DetectionNetworkProperties</a>
-------------------	---

**Methods:**

<code>__init__</code> (*args, **kwargs)	Initialize self.
<code>setConfidenceThreshold</code> (self, thresh)	Specifies confidence threshold at which to filter the rest of the detections.

**Attributes:**

<i>input</i>	Input message with data to be inferred upon Default queue is blocking with size 5
<i>out</i>	Outputs <i>ImgDetections</i> message that carries parsed detection results.
<i>passthrough</i>	Passthrough message on which the inference was performed.

**Properties**

alias of [depthai.DetectionNetworkProperties](#)

**Methods:**

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

**Attributes:**

<code>anchorMasks</code>
<code>anchors</code>
<code>classes</code>
<code>confidenceThreshold</code>
<code>coordinates</code>
<code>iouThreshold</code>
<code>nnFamily</code>



**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**property input**

Input message with data to be inferred upon Default queue is blocking with size 5

**property out**

Outputs ImgDetections message that carries parsed detection results.

**property passthrough**

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

**setConfidenceThreshold** (self: [depthai.DetectionNetwork](#), thresh: *float*) → *None*

Specifies confidence threshold at which to filter the rest of the detections.

**Parameter thresh:** Detection confidence must be greater than specified threshold to be added to the list

**class** [depthai.DetectionNetworkProperties](#)

Bases: [depthai.NeuralNetworkProperties](#)

Properties for DetectionNetwork

**Methods:**

---

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self.

---

**Attributes:**

---

[anchorMasks](#)

---

[anchors](#)

---

[classes](#)

---

[confidenceThreshold](#)

---

[coordinates](#)

---

[iouThreshold](#)

---

[nnFamily](#)

---

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**property anchorMasks**

**property anchors**

**property classes**

**property confidenceThreshold**

**property coordinates**

**property iouThreshold**

**property nnFamily**

**class** [depthai.Device](#)

Bases: [pybind11\\_builtins.pybind11\\_object](#)

Represents the DepthAI device with the methods to interact with it.

**Methods:**

<code>__init__(*args, **kwargs)</code>	Overloaded function.
<code>addLogCallback(self, callback)</code>	Add a callback for device logging.
<code>close(self)</code>	Closes the connection to device.
<code>getAllAvailableDevices()</code>	Returns all connected devices
<code>getAnyAvailableDevice(*args, **kwargs)</code>	Overloaded function.
<code>getChipTemperature(self)</code>	Retrieves current chip temperature as measured by device
<code>getCmxMemoryUsage(self)</code>	Retrieves current CMX memory information from device
<code>getDdrMemoryUsage(self)</code>	Retrieves current DDR memory information from device
<code>getDeviceByMxId(mxId)</code>	Finds a device by MX ID.
<code>getEmbeddedDeviceBinary(usb2Mode, version)</code>	Gets device firmware binary for a specific OpenVINO version
<code>getFirstAvailableDevice()</code>	Gets first available device.
<code>getInputQueue(*args, **kwargs)</code>	Overloaded function.
<code>getInputQueueNames(self)</code>	Get all available input queue names
<code>getLeonCssCpuUsage(self)</code>	Retrieves average CSS Leon CPU usage
<code>getLeonCssHeapUsage(self)</code>	Retrieves current CSS Leon CPU heap information from device
<code>getLeonMssCpuUsage(self)</code>	Retrieves average MSS Leon CPU usage
<code>getLeonMssHeapUsage(self)</code>	Retrieves current MSS Leon CPU heap information from device
<code>getLogLevel(self)</code>	Gets current logging severity level of the device.
<code>getLogOutputLevel(self)</code>	Gets logging level which decides printing level to standard output.
<code>getOutputQueue(*args, **kwargs)</code>	Overloaded function.
<code>getOutputQueueNames(self)</code>	Get all available output queue names
<code>getQueueEvent(*args, **kwargs)</code>	Overloaded function.
<code>getQueueEvents(*args, **kwargs)</code>	Overloaded function.
<code>getSystemInformationLoggingRate(self)</code>	Gets current rate of system information logging (“info” severity) in Hz.
<code>isPipelineRunning(self)</code>	Checks if devices pipeline is already running
<code>removeLogCallback(self, callbackId)</code>	Removes a callback
<code>setLogLevel(self, level)</code>	Sets the devices logging severity level.
<code>setLogOutputLevel(self, level)</code>	Sets logging level which decides printing level to standard output.
<code>setSystemInformationLoggingRate(self, rateHz)</code>	Sets rate of system information logging (“info” severity).
<code>startPipeline(self)</code>	Starts the execution of the devices pipeline

`__init__ (*args, **kwargs)`

Overloaded function.

1. `__init__(self: depthai.Device, pipeline: depthai.Pipeline) -> None`

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameter pipeline:**

- Pipeline to be executed on the device

2. `__init__(self: depthai.Device, pipeline: depthai.Pipeline, usb2Mode: bool) -> None`

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameter pipeline:**

- Pipeline to be executed on the device

**Parameter usb2Mode:**

- Boot device using USB2 mode firmware

3. `__init__(self: depthai.Device, pipeline: depthai.Pipeline, pathToCmd: str) -> None`

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameter pipeline:**

- Pipeline to be executed on the device

**Parameter pathToCmd:**

- Path to custom device firmware

4. `__init__(self: depthai.Device, pipeline: depthai.Pipeline, deviceDesc: depthai.DeviceInfo, usb2Mode: bool = False) -> None`

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameter pipeline:**

- Pipeline to be executed on the device

**Parameter pathToCmd:**

- Path to custom device firmware

5. `__init__(self: depthai.Device, pipeline: depthai.Pipeline, deviceDesc: depthai.DeviceInfo, pathToCmd: str) -> None`

Connects to device specified by devInfo.

**Parameter pipeline:**

- Pipeline to be executed on the device

**Parameter devInfo:**

- DeviceInfo which specifies which device to connect to

**Parameter usb2Mode:**

- Boot device using USB2 mode firmware

**addLogCallback** (*self: depthai.Device, callback: std::function<void (dai::LogMessage)>*) → `int`

Add a callback for device logging. The callback will be called from a separate thread with the LogMessage being passed.

**Parameter callback:**

- Callback to call whenever a log message arrives

**Returns** Id which can be used to later remove the callback

**close** (*self*: [depthai.Device](#)) → None

Closes the connection to device. Better alternative is the usage of context manager: *with depthai.Device(pipeline) as device:*

**static getAllAvailableDevices** () → List[[depthai.DeviceInfo](#)]

Returns all connected devices

**Returns** vector of connected devices

**static getAnyAvailableDevice** (\*args, \*\*kwargs)

Overloaded function.

1. `getAnyAvailableDevice(timeout: datetime.timedelta) -> Tuple[bool, depthai.DeviceInfo]`

Waits for any available device with a timeout

**Parameter timeout:**

- duration of time to wait for the any device

**Returns** a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

2. `getAnyAvailableDevice() -> Tuple[bool, depthai.DeviceInfo]`

Gets any available device

**Returns** a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

**getChipTemperature** (*self*: [depthai.Device](#)) → dai::ChipTemperature

Retrieves current chip temperature as measured by device

**Returns** Temperature of various onboard sensors

**getCmxMemoryUsage** (*self*: [depthai.Device](#)) → dai::MemoryInfo

Retrieves current CMX memory information from device

**Returns** Used, remaining and total cmx memory

**getDdrMemoryUsage** (*self*: [depthai.Device](#)) → dai::MemoryInfo

Retrieves current DDR memory information from device

**Returns** Used, remaining and total ddr memory

**static getDeviceByMxId** (*mxId*: str) → Tuple[bool, [depthai.DeviceInfo](#)]

Finds a device by MX ID. Example: 14442C10D13EABCE00

**Parameter mxId:**

- MyraidX ID which uniquely specifies a device

**Returns** a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

**static getEmbeddedDeviceBinary** (*usb2Mode*: bool, *version*: [depthai.OpenVINO.Version](#) = <[Version.VERSION\\_2020\\_1](#): 0>) → List[int]

Gets device firmware binary for a specific OpenVINO version

**Parameter usb2Mode:**

- USB2 mode firmware

**Parameter version:**

- Version of OpenVINO which firmware will support

**Returns** firmware binary

**static** `getFirstAvailableDevice()` → Tuple[bool, [depthai.DeviceInfo](#)]

Gets first available device. Device can be either in XLINK\_UNBOOTED or XLINK\_BOOTLOADER state

**Returns** a tuple of bool and DeviceInfo. Bool specifies if device was found. DeviceInfo specifies the found device

**getInputQueue** (\*args, \*\*kwargs)

Overloaded function.

1. `getInputQueue(self: depthai.Device, name: str) -> dai::DataInputQueue`

Gets an input queue corresponding to stream name. If it doesn't exist it throws

**Parameter name:** Queue/stream name, set in XLinkIn node

**Returns** Smart pointer to DataInputQueue

2. `getInputQueue(self: depthai.Device, name: str, maxSize: int, blocking: bool = True) -> dai::DataInputQueue`

Gets an input queue corresponding to stream name. If it doesn't exist it throws. Also sets queue options

**Parameter name:** Queue/stream name, set in XLinkOut node

**Parameter maxSize:** Maximum number of messages in queue

**Parameter blocking:** Queue behavior once full. True: blocking, false: overwriting of oldest messages. Default: true

**Returns** Smart pointer to DataInputQueue

**getInputQueueNames** (self: [depthai.Device](#)) → List[str]

Get all available input queue names

**Returns** Vector of input queue names

**getLeonCssCpuUsage** (self: [depthai.Device](#)) → dai::CpuUsage

Retrieves average CSS Leon CPU usage

**Returns** Average CPU usage and sampling duration

**getLeonCssHeapUsage** (self: [depthai.Device](#)) → dai::MemoryInfo

Retrieves current CSS Leon CPU heap information from device

**Returns** Used, remaining and total heap memory

**getLeonMssCpuUsage** (self: [depthai.Device](#)) → dai::CpuUsage

Retrieves average MSS Leon CPU usage

**Returns** Average CPU usage and sampling duration

**getLeonMssHeapUsage** (self: [depthai.Device](#)) → dai::MemoryInfo

Retrieves current MSS Leon CPU heap information from device

**Returns** Used, remaining and total heap memory

**getLogLevel** (self: [depthai.Device](#)) → dai::LogLevel

Gets current logging severity level of the device.

**Returns** Logging severity level

**getLogLevel**(*self*: [depthai.Device](#)) → [dai::LogLevel](#)

Gets logging level which decides printing level to standard output.

**Returns** Standard output printing severity

**getOutputQueue** (*\*args*, *\*\*kwargs*)

Overloaded function.

1. `getOutputQueue(self: depthai.Device, name: str) -> dai::DataOutputQueue`

Gets an output queue corresponding to stream name. If it doesn't exist it throws

**Parameter name:** Queue/stream name, created by XLinkOut node

**Returns** Smart pointer to DataOutputQueue

2. `getOutputQueue(self: depthai.Device, name: str, maxSize: int, blocking: bool = True) -> dai::DataOutputQueue`

Gets a queue corresponding to stream name, if it exists, otherwise it throws. Also sets queue options

**Parameter name:** Queue/stream name, set in XLinkOut node

**Parameter maxSize:** Maximum number of messages in queue

**Parameter blocking:** Queue behavior once full. True specifies blocking and false overwriting of oldest messages. Default: true

**Returns** Smart pointer to DataOutputQueue

**getOutputQueueNames** (*self*: [depthai.Device](#)) → [List\[str\]](#)

Get all available output queue names

**Returns** Vector of output queue names

**getQueueEvent** (*\*args*, *\*\*kwargs*)

Overloaded function.

1. `getQueueEvent(self: depthai.Device, queueNames: List[str], timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> str`

Gets or waits until any of specified queues has received a message

**Parameter queueNames:** Names of queues for which to wait for

**Parameter timeout:** Timeout after which return regardless. If negative then wait is indefinite. Default is -1

**Returns** Queue name which received a message first

2. `getQueueEvent(self: depthai.Device, queueName: str, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> str`

Gets or waits until specified queue has received a message

**Parameter queueNames:** Name of queues for which to wait for

**Parameter timeout:** Timeout after which return regardless. If negative then wait is indefinite. Default is -1

**Returns** Queue name which received a message

3. `getQueueEvent(self: depthai.Device, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> str`

Gets or waits until any queue has received a message

**Parameter timeout:** Timeout after which return regardless. If negative then wait is indefinite. Default is -1

**Returns** Queue name which received a message

**getQueueEvents** (\*args, \*\*kwargs)

Overloaded function.

1. `getQueueEvents(self: depthai.Device, queueNames: List[str], maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until any of specified queues has received a message

**Parameter queueNames:** Names of queues for which to block

**Parameter maxNumEvents:** Maximum number of events to remove from queue - Default is unlimited

**Parameter timeout:** Timeout after which return regardless. If negative then wait is indefinite - Default is -1

**Returns** Names of queues which received messages first

2. `getQueueEvents(self: depthai.Device, queueName: str, maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until specified queue has received a message

**Parameter queueName:** Name of queues for which to wait for

**Parameter maxNumEvents:** Maximum number of events to remove from queue. Default is unlimited

**Parameter timeout:** Timeout after which return regardless. If negative then wait is indefinite. Default is -1

**Returns** Names of queues which received messages first

3. `getQueueEvents(self: depthai.Device, maxNumEvents: int = 18446744073709551615, timeout: datetime.timedelta = datetime.timedelta(days=-1, seconds=86399, microseconds=999999)) -> List[str]`

Gets or waits until any any queue has received a message

**Parameter maxNumEvents:** Maximum number of events to remove from queue. Default is unlimited

**Parameter timeout:** Timeout after which return regardless. If negative then wait is indefinite. Default is -1

**Returns** Names of queues which received messages first

**getSystemInformationLoggingRate** (self: depthai.Device) → float

Gets current rate of system information logging ("info" severity) in Hz.

**Returns** Logging rate in Hz

**isPipelineRunning** (*self*: [depthai.Device](#)) → [bool](#)

Checks if devices pipeline is already running

**Returns** true if running, false otherwise

**removeLogCallback** (*self*: [depthai.Device](#), *callbackId*: [int](#)) → [bool](#)

Removes a callback

**Parameter callbackId:** Id of callback to be removed

**Returns** true if callback was removed, false otherwise

**setLogLevel** (*self*: [depthai.Device](#), *level*: [dai::LogLevel](#)) → [None](#)

Sets the devices logging severity level. This level affects which logs are transferred from device to host.

**Parameter level:** Logging severity

**setLogOutputLevel** (*self*: [depthai.Device](#), *level*: [dai::LogLevel](#)) → [None](#)

Sets logging level which decides printing level to standard output. If lower than setLogLevel, no messages will be printed

**Parameter level:**

- Standard output printing severity

**setSystemInformationLoggingRate** (*self*: [depthai.Device](#), *rateHz*: [float](#)) → [None](#)

Sets rate of system information logging (“info” severity). Default 1Hz If parameter is less or equal to zero, then system information logging will be disabled

**Parameter rateHz:** Logging rate in Hz

**startPipeline** (*self*: [depthai.Device](#)) → [bool](#)

Starts the execution of the devices pipeline

**Returns** true if pipeline started, false otherwise

**class** [depthai.DeviceBootloader](#)

Bases: [pybind11\\_builtins.pybind11\\_object](#)

Represents the DepthAI bootloader with the methods to interact with it.

**Classes:**

<i>Version</i>	Bootloader version structure
<b>Methods:</b>	
<code>__init__</code> (*args, **kwargs)	Overloaded function.
<code>close</code> (self)	Closes the connection to device.
<code>createDepthaiApplicationPackage</code> (pipeline, ...)	Creates application package which can be flashed to depthai device.
<code>flash</code> (self, progressCallback, None], pipeline)	Flashes a give pipeline to the board.
<code>flashBootloader</code> (self, progressCallback, ...)	Flashes bootloader to the current board
<code>flashDepthaiApplicationPackage</code> (self, ...)	Flashes a specific depthai application package that was generated using createDepthaiApplicationPackage or saveDepthaiApplicationPackage

continues on next page



Table 51 – continued from previous page

<code>getAllAvailableDevices()</code>	Searches for connected devices in either UN-BOOTED or BOOTLOADER states.
<code>getEmbeddedBootloaderBinary()</code>	<b>returns</b> Embedded bootloader binary
<code>getEmbeddedBootloaderVersion()</code>	<b>returns</b> Embedded bootloader version
<code>getFirstAvailableDevice()</code>	Searches for connected devices in either UN-BOOTED or BOOTLOADER states and returns first available.
<code>getVersion(self)</code>	<b>returns</b> Version of current running boot-loader
<code>isEmbeddedVersion(self)</code>	<b>returns</b> True whether the bootloader running is flashed or booted by library
<code>saveDepthaiApplicationPackage(path, ...)</code>	Saves application package to a file which can be flashed to depthai device.

**class Version**Bases: `pybind11_builtins.pybind11_object`

Bootloader version structure

**Methods:**

<code>__init__(*args, **kwargs)</code>	Overloaded function.
<code>__init__(*args, **kwargs)</code> Overloaded function. 1. <code>__init__(self: depthai.DeviceBootloader.Version, v: str) -&gt; None</code> Construct Version from string 2. <code>__init__(self: depthai.DeviceBootloader.Version, major: int, minor: int, patch: int) -&gt; None</code> Construct Version major, minor and patch numbers	
<code>__init__(*args, **kwargs)</code> Overloaded function. 1. <code>__init__(self: depthai.DeviceBootloader, deviceDesc: depthai.DeviceInfo) -&gt; None</code> 2. <code>__init__(self: depthai.DeviceBootloader, deviceDesc: depthai.DeviceInfo, pathToCmd: str) -&gt; None</code> Connects to or boots device in bootloader mode depending on devInfo state. <b>Parameter devInfo:</b> DeviceInfo of which to boot or connect to	
<b>close</b> ( <i>self</i> : <code>depthai.DeviceBootloader</code> ) $\rightarrow$ <code>None</code> Closes the connection to device. Better alternative is the usage of context manager: <i>with depthai.DeviceBootloader(deviceInfo) as bootloader:</i>	
<b>static createDepthaiApplicationPackage</b> ( <i>pipeline</i> : <code>depthai.Pipeline</code> , <i>pathToCmd</i> : <code>str</code> = <code>"</code> ) $\rightarrow$ <code>List[int]</code> Creates application package which can be flashed to depthai device.	

**Parameter pipeline:** Pipeline from which to create the application package

**Parameter pathToCmd:** Optional path to custom device firmware

**Returns** Depthai application package

**flash** (*self*: [depthai.DeviceBootloader](#), *progressCallback*: *Callable*[[*float*], *None*], *pipeline*: [depthai.Pipeline](#)) → *Tuple*[*bool*, *str*]  
Flashes a give pipeline to the board.

**Parameter progressCallback:** Callback that sends back a value between 0..1 which signifies current flashing progress

**Parameter pipeline:** Pipeline to flash to the board

**flashBootloader** (*self*: [depthai.DeviceBootloader](#), *progressCallback*: *Callable*[[*float*], *None*], *path*: *str* = "") → *Tuple*[*bool*, *str*]  
Flashes bootloader to the current board

**Parameter progressCallback:** Callback that sends back a value between 0..1 which signifies current flashing progress

**Parameter path:** Optional parameter to custom bootloader to flash

**flashDepthaiApplicationPackage** (*self*: [depthai.DeviceBootloader](#), *progressCallback*: *Callable*[[*float*], *None*], *package*: *List*[*int*]) → *Tuple*[*bool*, *str*]  
Flashes a specific depthai application package that was generated using `createDepthaiApplicationPackage` or `saveDepthaiApplicationPackage`

**Parameter progressCallback:** Callback that sends back a value between 0..1 which signifies current flashing progress

**Parameter package:** Depthai application package to flash to the board

**static getAllAvailableDevices** () → *List*[[depthai.DeviceInfo](#)]  
Searches for connected devices in either UNBOOTED or BOOTLOADER states.

**Returns** Vector of all found devices

**static getEmbeddedBootloaderBinary** () → *List*[*int*]

**Returns** Embedded bootloader binary

**static getEmbeddedBootloaderVersion** () → [depthai.DeviceBootloader.Version](#)

**Returns** Embedded bootloader version

**static getFirstAvailableDevice** () → *Tuple*[*bool*, [depthai.DeviceInfo](#)]  
Searches for connected devices in either UNBOOTED or BOOTLOADER states and returns first available.

**Returns** Tuple of boolean and DeviceInfo. If found boolean is true and DeviceInfo describes the device. Otherwise false

**getVersion** (*self*: [depthai.DeviceBootloader](#)) → [depthai.DeviceBootloader.Version](#)

**Returns** Version of current running bootloader

**isEmbeddedVersion** (*self*: [depthai.DeviceBootloader](#)) → *bool*

**Returns** True whether the bootloader running is flashed or booted by library

**static saveDepthaiApplicationPackage** (*path*: *str*, *pipeline*: [depthai.Pipeline](#), *pathToCmd*: *str* = "") → *None*  
Saves application package to a file which can be flashed to depthai device.

**Parameter path:** Path where to save the application package

**Parameter pipeline:** Pipeline from which to create the application package

**Parameter pathToCmd:** Optional path to custom device firmware

**class** depthai.DeviceDesc

Bases: pybind11\_builtins.pybind11\_object

**Methods:**

---

`__init__(self)`

---

**Attributes:**

---

`name`

---

`platform`

---

`protocol`

---

`__init__(self: depthai.DeviceDesc) → None`

**property** name

**property** platform

**property** protocol

**class** depthai.DeviceInfo

Bases: pybind11\_builtins.pybind11\_object

**Methods:**

---

`__init__(self)`

---

`getMxId(self)`

---

**Attributes:**

---

`desc`

---

`state`

---

`__init__(self: depthai.DeviceInfo) → None`

**property** desc

`getMxId(self: depthai.DeviceInfo) → str`

**property** state

**class** depthai.GlobalProperties

Bases: pybind11\_builtins.pybind11\_object

Specify properties which apply for whole pipeline

**Methods:**

---

`__init__(*args, **kwargs)`

Initialize self.

---

**Attributes:**

---

*leonOsFrequencyHz*

---

*leonRtFrequencyHz*

---

*pipelineName*

---

*pipelineVersion*

---

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**property** *leonOsFrequencyHz***property** *leonRtFrequencyHz***property** *pipelineName***property** *pipelineVersion***class** *depthai.ImageManip*Bases: *depthai.Node*

ImageManip node. Capability to crop, resize, warp, ... incoming image frames

**Methods:**

---

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self.

*setCenterCrop*(self, arg0, arg1)*setCropRect*(self, arg0, arg1, arg2, arg3) *setFrameType*(self, arg0)*setHorizontalFlip*(self, arg0)*setKeepAspectRatio*(self, arg0)*setMaxOutputFrameSize*(self, arg0)

Specify maximum size of output image.

*setNumFramesPool*(self, arg0)

Specify number of frames in pool.

*setResize*(self, arg0, arg1)*setResizeThumbnail*(self, arg0, arg1, arg2,

...)

*setWaitForConfigInput*(self, arg0)

Specify whether or not wait until configuration message arrives to inputConfig Input.

**Attributes:**

---

*initialConfig*

Initial config to use when manipulating frames

*inputConfig*

Input ImageManipConfig message with ability to modify parameters in runtime Default queue is blocking with size 8

*inputImage*

Input image to be modified Default queue is blocking with size 8

*out*

Outputs ImgFrame message that carries modified image.

---

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**property** *initialConfig*

Initial config to use when manipulating frames

**property** *inputConfig*

Input ImageManipConfig message with ability to modify parameters in runtime Default queue is blocking

with size 8

**property inputImage**

Input image to be modified Default queue is blocking with size 8

**property out**

Outputs ImgFrame message that carries modified image.

**setCenterCrop** (*self*: depthai.ImageManip, *arg0*: float, *arg1*: float) → None

**setCropRect** (*self*: depthai.ImageManip, *arg0*: float, *arg1*: float, *arg2*: float, *arg3*: float) → None

**setFrameType** (*self*: depthai.ImageManip, *arg0*: dai::RawImgFrame::Type) → None

**setHorizontalFlip** (*self*: depthai.ImageManip, *arg0*: bool) → None

**setKeepAspectRatio** (*self*: depthai.ImageManip, *arg0*: bool) → None

**setMaxOutputFrameSize** (*self*: depthai.ImageManip, *arg0*: int) → None

Specify maximum size of output image.

**Parameter maxFrameSize:** Maximum frame size in bytes

**setNumFramesPool** (*self*: depthai.ImageManip, *arg0*: int) → None

Specify number of frames in pool.

**Parameter numFramesPool:** How many frames should the pool have

**setResize** (*self*: depthai.ImageManip, *arg0*: int, *arg1*: int) → None

**setResizeThumbnail** (*self*: depthai.ImageManip, *arg0*: int, *arg1*: int, *arg2*: int, *arg3*: int, *arg4*: int) → None

**setWaitForConfigInput** (*self*: depthai.ImageManip, *arg0*: bool) → None

Specify whether or not wait until configuration message arrives to inputConfig Input.

**Parameter wait:** True to wait for configuration message, false otherwise

**class depthai.ImageManipConfig**

Bases: *depthai.Buffer*

ImageManipConfig message. Specifies image manipulation options like:

- Crop
- Resize
- Warp
- ...

**Methods:**

---

*\_\_init\_\_*(self)

---

*getCropXMax*(self)

---

**returns** Bottom right X coordinate of crop region

---

*getCropXMin*(self)

---

**returns** Top left X coordinate of crop region

---

continues on next page

Table 61 – continued from previous page

<code>getCropYMax(self)</code>	<b>returns</b> Bottom right Y coordinate of crop region
<code>getCropYMin(self)</code>	<b>returns</b> Top left Y coordinate of crop region
<code>getResizeHeight(self)</code>	<b>returns</b> Output image height
<code>getResizeWidth(self)</code>	<b>returns</b> Output image width
<code>isResizeThumbnail(self)</code>	<b>returns</b> True if resize thumbnail mode is set, false otherwise
<code>setCenterCrop(self, ratio, whRatio)</code>	Specifies a centered crop.
<code>setCropRect(self, xmin, ymin, xmax, ymax)</code>	Specifies crop with rectangle with normalized values (0..1)
<code>setCropRotatedRect(self, rr, normalizedCoords)</code>	Specifies crop with rotated rectangle.
<code>setFrameType(self, name)</code>	Specify output frame type.
<code>setHorizontalFlip(self, flip)</code>	Specify horizontal flip
<code>setKeepAspectRatio(self, keep)</code>	Specifies to whether to keep aspect ratio or not
<code>setResize(self, w, h)</code>	Specifies output image size.
<code>setResizeThumbnail(self, w, h, bgRed, ...)</code>	Specifies output image size.
<code>setReusePreviousImage(self, reuse)</code>	Instruct ImageManip to not remove current image from its queue and use the same for next message.
<code>setRotationDegrees(self, deg)</code>	Specifies clockwise rotation in degrees
<code>setRotationRadians(self, rad)</code>	Specifies clockwise rotation in radians
<code>setSkipCurrentImage(self, skip)</code>	Instructs ImageManip to skip current image and wait for next in queue.
<code>setWarpBorderFillColor(self, red, green, blue)</code>	Specifies fill color for border pixels.
<code>setWarpBorderReplicatePixels(self)</code>	Specifies that warp replicates border pixels
<code>setWarpTransformFourPoints(self, pt, ...)</code>	Specifies warp by supplying 4 points in either absolute or normalized coordinates
<code>setWarpTransformMatrix3x3(self, mat)</code>	Specifies warp with a 3x3 matrix

`__init__(self: depthai.ImageManipConfig) → None`

`getCropXMax(self: depthai.ImageManipConfig) → float`

**Returns** Bottom right X coordinate of crop region

`getCropXMin(self: depthai.ImageManipConfig) → float`

**Returns** Top left X coordinate of crop region

`getCropYMax(self: depthai.ImageManipConfig) → float`

**Returns** Bottom right Y coordinate of crop region

`getCropYMin(self: depthai.ImageManipConfig) → float`

**Returns** Top left Y coordinate of crop region

**getResizeHeight** (*self*: depthai.ImageManipConfig) → int

**Returns** Output image height

**getResizeWidth** (*self*: depthai.ImageManipConfig) → int

**Returns** Output image width

**isResizeThumbnail** (*self*: depthai.ImageManipConfig) → bool

**Returns** True if resize thumbnail mode is set, false otherwise

**setCenterCrop** (*self*: depthai.ImageManipConfig, *ratio*: float, *whRatio*: float = 1.0) → None

Specifies a centered crop.

**Parameter ratio:** Ratio between input image and crop region (0..1)

**Parameter whRatio:** Crop region aspect ratio - 1 equals to square, 1.7 equals to 16:9, ...

**ImageManipConfig.setCropRect** (*self*: depthai.ImageManipConfig, *xmin*: float, *ymin*: float,

Specifies crop with rectangle with normalized values (0..1)

**Parameter xmin:** Top left X coordinate of rectangle

**Parameter ymin:** Top left Y coordinate of rectangle

**Parameter xmax:** Bottom right X coordinate of rectangle

**Parameter ymax:** Bottom right Y coordinate of rectangle

**setCropRotatedRect** (*self*: depthai.ImageManipConfig, *rr*: depthai.RotatedRect, *normalizedCo-*  
*ords*: bool = True) → None

Specifies crop with rotated rectangle. Optionally as non normalized coordinates

**Parameter rr:** Rotated rectangle which specifies crop

**Parameter normalizedCoords:** If true coordinates are in normalized range (0..1) otherwise absolute

**setFrameType** (*self*: depthai.ImageManipConfig, *name*: depthai.RawImgFrame.Type) → None

Specify output frame type.

**Parameter name:** Frame type

**setHorizontalFlip** (*self*: depthai.ImageManipConfig, *flip*: bool) → None

Specify horizontal flip

**Parameter flip:** True to enable flip, false otherwise

**setKeepAspectRatio** (*self*: depthai.ImageManipConfig, *keep*: bool) → None

Specifies to whether to keep aspect ratio or not

**setResize** (*self*: depthai.ImageManipConfig, *w*: int, *h*: int) → None

Specifies output image size. After crop stage the image will be stretched to fit.

**Parameter w:** Width in pixels

**Parameter h:** Height in pixels

**setResizeThumbnail** (*self*: depthai.ImageManipConfig, *w*: int, *h*: int, *bgRed*: int = 0, *bgGreen*: int  
= 0, *bgBlue*: int = 0) → None

Specifies output image size. After crop stage the image will be resized by preserving aspect ration. Optionally background can be specified.

**Parameter w:** Width in pixels

**Parameter h:** Height in pixels

**Parameter bgRed:** Red component

**Parameter bgGreen:** Green component

**Parameter bgBlue:** Blue component

**setReusePreviousImage** (*self*: [depthai.ImageManipConfig](#), *reuse*: *bool*) → *None*

Instruct ImageManip to not remove current image from its queue and use the same for next message.

**Parameter reuse:** True to enable reuse, false otherwise

**setRotationDegrees** (*self*: [depthai.ImageManipConfig](#), *deg*: *float*) → *None*

Specifies clockwise rotation in degrees

**Parameter deg:** Rotation in degrees

**setRotationRadians** (*self*: [depthai.ImageManipConfig](#), *rad*: *float*) → *None*

Specifies clockwise rotation in radians

**Parameter rad:** Rotation in radians

**setSkipCurrentImage** (*self*: [depthai.ImageManipConfig](#), *skip*: *bool*) → *None*

Instructs ImageManip to skip current image and wait for next in queue.

**Parameter skip:** True to skip current image, false otherwise

**setWarpBorderFillColor** (*self*: [depthai.ImageManipConfig](#), *red*: *int*, *green*: *int*, *blue*: *int*) → *None*

Specifies fill color for border pixels. Example:

- `setWarpBorderFillColor(255,255,255)` -> white
- `setWarpBorderFillColor(0,0,255)` -> blue

**Parameter red:** Red component

**Parameter green:** Green component

**Parameter blue:** Blue component

**setWarpBorderReplicatePixels** (*self*: [depthai.ImageManipConfig](#)) → *None*

Specifies that warp replicates border pixels

**setWarpTransformFourPoints** (*self*: [depthai.ImageManipConfig](#), *pt*: *List[depthai.Point2f]*, *normalizedCoords*: *bool*) → *None*

Specifies warp by suppling 4 points in either absolute or normalized coordinates

**Parameter pt:** 4 points specifying warp

**Parameter normalizedCoords:** If true pt is interpreted as normalized, absolute otherwise

**setWarpTransformMatrix3x3** (*self*: [depthai.ImageManipConfig](#), *mat*: *List[float]*) → *None*

Specifies warp with a 3x3 matrix

**Parameter mat:** 3x3 matrix

**class** [depthai.ImgDetection](#)

Bases: [pybind11\\_builtins.pybind11\\_object](#)

**Methods:**

---

`\_\_init\_\_`(*self*)

---

**Attributes:**



---

*confidence*

---

*label*

---

*xmax*

---

*xmin*

---

*ymax*

---

*ymin*

---

```
__init__(self: depthai.ImgDetection) → None
```

```
property confidence
```

```
property label
```

```
property xmax
```

```
property xmin
```

```
property ymax
```

```
property ymin
```

```
class depthai.ImgDetections
```

```
Bases: depthai.Buffer
```

ImgDetections message. Carries normalized detection results

**Methods:**

---

```
__init__(self)
```

---

Construct ImgDetections message

**Attributes:**

---

*detections*

---

Detections

```
__init__(self: depthai.ImgDetections) → None
```

```
Construct ImgDetections message
```

```
property detections
```

```
Detections
```

```
class depthai.ImgFrame
```

```
Bases: depthai.Buffer
```

ImgFrame message. Carries image data and metadata.

**Classes:**

---

*Specs*

---

*Type*

---

Members:

**Methods:**

---

```
__init__(self)
```

---

```
getCategory(self)
```

---

Retrieves image category

```
getCvFrame(self)
```

---

Returns BGR or grayscale frame compatible with use in other opencv functions

continues on next page

Table 67 – continued from previous page

<code>getFrame(self, copy)</code>	Returns numpy array with shape as specified by width, height and type
<code>getHeight(self)</code>	Retrieves image height in pixels
<code>getInstanceNum(self)</code>	Retrieves instance number
<code>getSequenceNum(self)</code>	Retrieves image sequence number
<code>getTimestamp(self)</code>	Retrieves image timestamp related to steady_clock / time.monotonic
<code>getType(self)</code>	Retrieves image type
<code>getWidth(self)</code>	Retrieves image width in pixels
<code>setCategory(self, category)</code>	Parameter category:
<code>setFrame(self, array)</code>	Copies array bytes to ImgFrame buffer
<code>setHeight(self, height)</code>	Specifies frame height
<code>setInstanceNum(self, instance)</code>	Instance number relates to the origin of the frame (which camera)
<code>setSequenceNum(self, seq)</code>	Specifies sequence number
<code>setTimestamp(self, timestamp)</code>	Specifies current timestamp, related to steady_clock / time.monotonic
<code>setType(self, type)</code>	Specifies frame type, RGB, BGR, ...
<code>setWidth(self, width)</code>	Specifies frame width

**class Specs**Bases: `pybind11_builtins.pybind11_object`**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<code>bytesPP</code>
<code>height</code>
<code>p1Offset</code>
<code>p2Offset</code>
<code>p3Offset</code>
<code>stride</code>
<code>type</code>
<code>width</code>

`__init__(*args, **kwargs)`Initialize self. See `help(type(self))` for accurate signature.**property bytesPP****property height****property p1Offset****property p2Offset****property p3Offset****property stride****property type**

**property width**

**class Type**

Bases: `pybind11_builtins.pybind11_object`

Members:

YUV422i

YUV444p

YUV420p

YUV422p

YUV400p

RGBA8888

RGB161616

RGB888p

BGR888p

RGB888i

BGR888i

RGBF16F16F16p

BGRF16F16F16p

RGBF16F16F16i

BGRF16F16F16i

GRAY8

GRAYF16

LUT2

LUT4

LUT16

RAW16

RAW14

RAW12

RAW10

RAW8

PACK10

PACK12

YUV444i

NV12

NV21

BITSTREAM

HDR

NONE

**Attributes:**

<i>BGR888i</i>
<i>BGR888p</i>
<i>BGRF16F16F16i</i>
<i>BGRF16F16F16p</i>
<i>BITSTREAM</i>
<i>GRAY8</i>
<i>GRAYF16</i>
<i>HDR</i>
<i>LUT16</i>
<i>LUT2</i>
<i>LUT4</i>
<i>NONE</i>
<i>NV12</i>
<i>NV21</i>
<i>PACK10</i>
<i>PACK12</i>
<i>RAW10</i>
<i>RAW12</i>
<i>RAW14</i>
<i>RAW16</i>
<i>RAW8</i>
<i>RGB161616</i>
<i>RGB888i</i>
<i>RGB888p</i>
<i>RGBA8888</i>
<i>RGBF16F16F16i</i>
<i>RGBF16F16F16p</i>
<i>YUV400p</i>
<i>YUV420p</i>
<i>YUV422i</i>
<i>YUV422p</i>
<i>YUV444i</i>
<i>YUV444p</i>
<i>name</i>
<i>value</i>

**Methods:**

<code>__init__(self, value)</code>
------------------------------------

**BGR888i** = <Type.BGR888i: 10>

**BGR888p** = <Type.BGR888p: 8>

**BGRF16F16F16i** = <Type.BGRF16F16F16i: 14>

**BGRF16F16F16p** = <Type.BGRF16F16F16p: 12>

**BITSTREAM** = <Type.BITSTREAM: 30>

**GRAY8** = <Type.GRAY8: 15>

```

GRAYF16 = <Type.GRAYF16: 16>
HDR = <Type.HDR: 31>
LUT16 = <Type.LUT16: 19>
LUT2 = <Type.LUT2: 17>
LUT4 = <Type.LUT4: 18>
NONE = <Type.NONE: 32>
NV12 = <Type.NV12: 28>
NV21 = <Type.NV21: 29>
PACK10 = <Type.PACK10: 25>
PACK12 = <Type.PACK12: 26>
RAW10 = <Type.RAW10: 23>
RAW12 = <Type.RAW12: 22>
RAW14 = <Type.RAW14: 21>
RAW16 = <Type.RAW16: 20>
RAW8 = <Type.RAW8: 24>
RGB161616 = <Type.RGB161616: 6>
RGB888i = <Type.RGB888i: 9>
RGB888p = <Type.RGB888p: 7>
RGBA8888 = <Type.RGBA8888: 5>
RGBF16F16F16i = <Type.RGBF16F16F16i: 13>
RGBF16F16F16p = <Type.RGBF16F16F16p: 11>
YUV400p = <Type.YUV400p: 4>
YUV420p = <Type.YUV420p: 2>
YUV422i = <Type.YUV422i: 0>
YUV422p = <Type.YUV422p: 3>
YUV444i = <Type.YUV444i: 27>
YUV444p = <Type.YUV444p: 1>

__init__(self: depthai.RawImgFrame.Type, value: int) → None
property name
property value

__init__(self: depthai.ImgFrame) → None
getCategory(self: depthai.ImgFrame) → int
    Retrieves image category
getCvFrame(self: object) → object
    Returns BGR or grayscale frame compatible with use in other opencv functions
getFrame(self: object, copy: bool = False) → numpy.ndarray
    Returns numpy array with shape as specified by width, height and type

```

**getHeight** (*self*: `depthai.ImgFrame`) → `int`  
Retrieves image height in pixels

**getInstanceNum** (*self*: `depthai.ImgFrame`) → `int`  
Retrieves instance number

**getSequenceNum** (*self*: `depthai.ImgFrame`) → `int`  
Retrieves image sequence number

**getTimestamp** (*self*: `depthai.ImgFrame`) → `datetime.timedelta`  
Retrieves image timestamp related to `steady_clock` / `time.monotonic`

**getType** (*self*: `depthai.ImgFrame`) → `depthai.RawImgFrame.Type`  
Retrieves image type

**getWidth** (*self*: `depthai.ImgFrame`) → `int`  
Retrieves image width in pixels

**setCategory** (*self*: `depthai.ImgFrame`, *category*: `int`) → `None`  
**Parameter category:** Image category

**setFrame** (*self*: `depthai.ImgFrame`, *array*: `numpy.ndarray`) → `None`  
Copies array bytes to `ImgFrame` buffer

**setHeight** (*self*: `depthai.ImgFrame`, *height*: `int`) → `None`  
Specifies frame height  
**Parameter width:** frame height

**setInstanceNum** (*self*: `depthai.ImgFrame`, *instance*: `int`) → `None`  
Instance number relates to the origin of the frame (which camera)  
**Parameter instance:** Instance number

**setSequenceNum** (*self*: `depthai.ImgFrame`, *seq*: `int`) → `None`  
Specifies sequence number  
**Parameter seq:** Sequence number

**setTimestamp** (*self*: `depthai.ImgFrame`, *timestamp*: `datetime.timedelta`) → `None`  
Specifies current timestamp, related to `steady_clock` / `time.monotonic`

**setType** (*self*: `depthai.ImgFrame`, *type*: `depthai.RawImgFrame.Type`) → `None`  
Specifies frame type, RGB, BGR, ...  
**Parameter type:** Type of image

**setWidth** (*self*: `depthai.ImgFrame`, *width*: `int`) → `None`  
Specifies frame width  
**Parameter width:** frame width

**class** `depthai.LogLevel`  
Bases: `pybind11_builtins.pybind11_object`  
Members:  
TRACE  
DEBUG  
INFO  
WARN  
ERR

CRITICAL

OFF

**Attributes:**

---

*CRITICAL*

---

*DEBUG*

---

*ERR*

---

*INFO*

---

*OFF*

---

*TRACE*

---

*WARN*

---

*name*

---

*value*

---

**Methods:**

---

*\_\_init\_\_*(self, value)

---

**CRITICAL** = <LogLevel.CRITICAL: 5>

**DEBUG** = <LogLevel.DEBUG: 1>

**ERR** = <LogLevel.ERR: 4>

**INFO** = <LogLevel.INFO: 2>

**OFF** = <LogLevel.OFF: 6>

**TRACE** = <LogLevel.TRACE: 0>

**WARN** = <LogLevel.WARN: 3>

*\_\_init\_\_* (self: depthai.LogLevel, value: int) → None

**property name**

**property value**

**class** depthai.MemoryInfo

Bases: pybind11\_builtins.pybind11\_object

MemoryInfo structure

Free, remaining and total memory stats

**Methods:**

---

*\_\_init\_\_*(self)

---

**Attributes:**

---

*remaining*

---

*total*

---

*used*

---

*\_\_init\_\_* (self: depthai.MemoryInfo) → None

**property** remaining

**property** total

**property** used

**class** depthai.MobileNetDetectionNetwork

Bases: *depthai.DetectionNetwork*

MobileNetDetectionNetwork node. Parses MobileNet results

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

`__init__(*args, **kwargs)`  
 Initialize self. See help(type(self)) for accurate signature.

**class** depthai.MobileNetSpatialDetectionNetwork

Bases: *depthai.SpatialDetectionNetwork*

MobileNetSpatialDetectionNetwork. Mobilenet-SSD based network with spatial location data.

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

`__init__(*args, **kwargs)`  
 Initialize self. See help(type(self)) for accurate signature.

**class** depthai.MonoCamera

Bases: *depthai.Node*

MonoCamera node. For use with grayscale sensors.

**Classes:**

<i>Properties</i>	alias of <i>depthai.MonoCameraProperties</i>
-------------------	--

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBoardSocket(self)</code>	Retrieves which board socket to use
<code>getCamId(self)</code>	
<code>getFps(self)</code>	Get rate at which camera should produce frames
<code>getImageOrientation(self)</code>	Get camera image orientation
<code>getResolution(self)</code>	Get sensor resolution
<code>getResolutionHeight(self)</code>	Get sensor resolution height
<code>getResolutionSize(self)</code>	Get sensor resolution as size
<code>getResolutionWidth(self)</code>	Get sensor resolution width
<code>setBoardSocket(self, boardSocket)</code>	Specify which board socket to use
<code>setCamId(self, arg0)</code>	
<code>setFps(self, fps)</code>	Set rate at which camera should produce frames
<code>setImageOrientation(self, imageOrientation)</code>	Set camera image orientation
<code>setResolution(self, resolution)</code>	Set sensor resolution

**Attributes:**



<i>initialControl</i>	Initial control options to apply to sensor
<i>inputControl</i>	Input for CameraControl message, which can modify camera parameters in runtime Default queue is blocking with size 8
<i>out</i>	Outputs ImgFrame message that carries RAW8 encoded (grayscale) frame data.

**Properties**alias of *depthai.MonoCameraProperties* **Classes:**

SensorResolution	Select the camera sensor resolution: 1280×720, 1280×800, 640×400
------------------	--

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

boardSocket
fps
initialControl
resolution

`__init__(*args, **kwargs)`  
Initialize self. See help(type(self)) for accurate signature.

**getBoardSocket** (*self*: [depthai.MonoCamera](#)) → [dai::CameraBoardSocket](#)  
Retrieves which board socket to use

**Returns** Board socket to use

**getCamId** (*self*: [depthai.MonoCamera](#)) → [int](#)

**getFps** (*self*: [depthai.MonoCamera](#)) → [float](#)  
Get rate at which camera should produce frames

**Returns** Rate in frames per second

**getImageOrientation** (*self*: [depthai.MonoCamera](#)) → [dai::CameraImageOrientation](#)  
Get camera image orientation

**getResolution** (*self*: [depthai.MonoCamera](#)) → [dai::MonoCameraProperties::SensorResolution](#)  
Get sensor resolution

**getResolutionHeight** (*self*: [depthai.MonoCamera](#)) → [int](#)  
Get sensor resolution height

**getResolutionSize** (*self*: [depthai.MonoCamera](#)) → [Tuple\[int, int\]](#)  
Get sensor resolution as size

**getResolutionWidth** (*self*: [depthai.MonoCamera](#)) → [int](#)  
Get sensor resolution width

**property initialControl**  
Initial control options to apply to sensor

**property inputControl**

Input for CameraControl message, which can modify camera parameters in runtime Default queue is blocking with size 8

**property out**

Outputs ImgFrame message that carries RAW8 encoded (grayscale) frame data.

Suitable for use StereoDepth node

**setBoardSocket** (*self*: *depthai.MonoCamera*, *boardSocket*: *dai::CameraBoardSocket*) → *None*

Specify which board socket to use

**Parameter boardSocket:** Board socket to use

**setCamId** (*self*: *depthai.MonoCamera*, *arg0*: *int*) → *None*

**setFps** (*self*: *depthai.MonoCamera*, *fps*: *float*) → *None*

Set rate at which camera should produce frames

**Parameter fps:** Rate in frames per second

**setImageOrientation** (*self*: *depthai.MonoCamera*, *imageOrientation*: *dai::CameraImageOrientation*) → *None*

Set camera image orientation

**setResolution** (*self*: *depthai.MonoCamera*, *resolution*: *dai::MonoCameraProperties::SensorResolution*) → *None*

Set sensor resolution

**class depthai.MonoCameraProperties**

Bases: *pybind11\_builtins.pybind11\_object*

Specify MonoCamera options such as camera ID, ...

**Classes:**

---

<i>SensorResolution</i>	Select the camera sensor resolution: 1280×720, 1280×800, 640×400
-------------------------	--

---

**Methods:**

---

<i>__init__</i> (*args, **kwargs)	Initialize self.
-----------------------------------	------------------

---

**Attributes:**

---

<i>boardSocket</i>
<i>fps</i>
<i>initialControl</i>
<i>resolution</i>

---

**class SensorResolution**

Bases: *pybind11\_builtins.pybind11\_object*

Select the camera sensor resolution: 1280×720, 1280×800, 640×400

Members:

THE\_720\_P

THE\_800\_P

THE\_400\_P

**Attributes:**

<i>THE_400_P</i>
<i>THE_720_P</i>
<i>THE_800_P</i>
<i>name</i>
<i>value</i>

**Methods:**

<i>__init__</i> (self, value)
-------------------------------

**THE\_400\_P** = <SensorResolution.THE\_400\_P: 2>

**THE\_720\_P** = <SensorResolution.THE\_720\_P: 0>

**THE\_800\_P** = <SensorResolution.THE\_800\_P: 1>

*\_\_init\_\_*(self: depthai.MonoCameraProperties.SensorResolution, value: int) → None

**property name**

**property value**

*\_\_init\_\_*(\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**property boardSocket**

**property fps**

**property initialControl**

**property resolution**

**class** depthai.NNData

Bases: *depthai.Buffer*

NNData message. Carries tensors and their metadata

**Methods:**

<i>__init__</i> (self)	Construct NNData message.
<i>getAllLayerNames</i> (self)	<b>returns</b> Names of all layers added
<i>getAllLayers</i> (self)	<b>returns</b> All layers and their information
<i>getFirstLayerFp16</i> (self)	Convenience function to retrieve float values from first layers FP16 tensor
<i>getFirstLayerInt32</i> (self)	Convenience function to retrieve INT32 values from first layers tensor
<i>getFirstLayerUInt8</i> (self)	Convenience function to retrieve U8 data from first layer
<i>getLayer</i> (self, name, tensor)	Retrieve layers tensor information

continues on next page

Table 89 – continued from previous page

<code>getLayerDatatype(self, name, datatype)</code>	Retrieve datatype of a layers tensor
<code>getLayerFp16(self, name)</code>	Convenience function to retrieve float values from layers FP16 tensor
<code>getLayerInt32(self, name)</code>	Convenience function to retrieve INT32 values from layers tensor
<code>getLayerUInt8(self, name)</code>	Convenience function to retrieve U8 data from layer
<code>hasLayer(self, name)</code>	Checks if given layer exists
<code>setLayer(*args, **kwargs)</code>	Overloaded function.

`__init__` (*self*: `depthai.NNData`) → `None`

Construct NNData message.

`getAllLayerNames` (*self*: `depthai.NNData`) → `List[str]`

**Returns** Names of all layers added

`getAllLayers` (*self*: `depthai.NNData`) → `List[depthai.TensorInfo]`

**Returns** All layers and their information

`getFirstLayerFp16` (*self*: `depthai.NNData`) → `List[float]`

Convenience function to retrieve float values from first layers FP16 tensor

**Returns** Float data

`getFirstLayerInt32` (*self*: `depthai.NNData`) → `List[int]`

Convenience function to retrieve INT32 values from first layers tensor

**Returns** INT32 data

`getFirstLayerUInt8` (*self*: `depthai.NNData`) → `List[int]`

Convenience function to retrieve U8 data from first layer

**Returns** U8 binary data

`getLayer` (*self*: `depthai.NNData`, *name*: `str`, *tensor*: `depthai.TensorInfo`) → `bool`

Retrieve layers tensor information

**Parameter name:** Name of the layer

**Parameter tensor:** Outputs tensor information of that layer

**Returns** True if layer exists, false otherwise

`getLayerDatatype` (*self*: `depthai.NNData`, *name*: `str`, *datatype*: `depthai.TensorInfo.DataType`) → `bool`

Retrieve datatype of a layers tensor

**Parameter name:** Name of the layer

**Parameter datatype:** Datatype of layers tensor

**Returns** True if layer exists, false otherwise

`getLayerFp16` (*self*: `depthai.NNData`, *name*: `str`) → `List[float]`

Convenience function to retrieve float values from layers FP16 tensor

**Parameter name:** Name of the layer

**Returns** Float data

**getLayerInt32** (*self*: [depthai.NNData](#), *name*: *str*) → List[int]  
 Convenience function to retrieve INT32 values from layers tensor

**Parameter name:** Name of the layer

**Returns** INT32 data

**getLayerUInt8** (*self*: [depthai.NNData](#), *name*: *str*) → List[int]  
 Convenience function to retrieve U8 data from layer

**Parameter name:** Name of the layer

**Returns** U8 binary data

**hasLayer** (*self*: [depthai.NNData](#), *name*: *str*) → bool  
 Checks if given layer exists

**Parameter name:** Name of the layer

**Returns** True if layer exists, false otherwise

**setLayer** (*\*args*, *\*\*kwargs*)  
 Overloaded function.

1. `setLayer(self: depthai.NNData, name: str, data: numpy.ndarray[numpy.uint8]) -> None`

Set a layer with datatype U8.

**Parameter name:** Name of the layer

**Parameter data:** Data to store

2. `setLayer(self: depthai.NNData, name: str, data: List[int]) -> None`

Set a layer with datatype U8. Integers are casted to bytes.

**Parameter name:** Name of the layer

**Parameter data:** Data to store

3. `setLayer(self: depthai.NNData, name: str, data: List[float]) -> None`

Set a layer with datatype FP16. Float values are converted to FP16.

**Parameter name:** Name of the layer

**Parameter data:** Data to store

4. `setLayer(self: depthai.NNData, name: str, data: List[float]) -> None`

Set a layer with datatype FP16. Double values are converted to FP16.

**Parameter name:** Name of the layer

**Parameter data:** Data to store

**class** `depthai.NeuralNetwork`

Bases: [depthai.Node](#)

NeuralNetwork node. Runs a neural inference on input data.

**Classes:**

<i>Properties</i>	alias of <i>depthai.NeuralNetworkProperties</i>
-------------------	---

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getNumInferenceThreads(self)</code>	How many inference threads will be used to run the network
<code>setBlobPath(self, path)</code>	Load network blob into assets and use once pipeline is started.
<code>setNumInferenceThreads(self, numThreads)</code>	How many threads should the node use to run the network.
<code>setNumNCEPerInferenceThread(self, ...)</code>	How many Neural Compute Engines should a single thread use for inference
<code>setNumPoolFrames(self, numFrames)</code>	Specifies how many frames will be available in the pool

**Attributes:**

<i>input</i>	Input message with data to be inferred upon Default queue is blocking with size 5
<i>out</i>	Outputs NNData message that carries inference results
<i>passthrough</i>	Passthrough message on which the inference was performed.

**Properties**alias of *depthai.NeuralNetworkProperties* **Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<i>blobSize</i>
<i>blobUri</i>
<i>numFrames</i>
<i>numNCEPerThread</i>
<i>numThreads</i>

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**getNumInferenceThreads** (*self*: *depthai.NeuralNetwork*) → int

How many inference threads will be used to run the network

**Returns** Number of threads, 0, 1 or 2. Zero means AUTO**property input**

Input message with data to be inferred upon Default queue is blocking with size 5

**property out**

Outputs NNData message that carries inference results

**property passthrough**

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

**setBlobPath** (*self*: [depthai.NeuralNetwork](#), *path*: *str*) → [None](#)

Load network blob into assets and use once pipeline is started.

Throws if file doesn't exist or isn't a valid network blob.

**Parameter path:** Path to network blob

**setNumInferenceThreads** (*self*: [depthai.NeuralNetwork](#), *numThreads*: *int*) → [None](#)

How many threads should the node use to run the network.

**Parameter numThreads:** Number of threads to dedicate to this node

**setNumNCEPerInferenceThread** (*self*: [depthai.NeuralNetwork](#), *numNCEPerThread*: *int*) → [None](#)

How many Neural Compute Engines should a single thread use for inference

**Parameter numNCEPerThread:** Number of NCE per thread

**setNumPoolFrames** (*self*: [depthai.NeuralNetwork](#), *numFrames*: *int*) → [None](#)

Specifies how many frames will be available in the pool

**Parameter numFrames:** How many frames will pool have

**class** [depthai.NeuralNetworkProperties](#)

Bases: [pybind11\\_builtins.pybind11\\_object](#)

Specify NeuralNetwork options such as blob path, ...

**Methods:**


---

<code><a href="#">__init__</a>(*args, **kwargs)</code>	Initialize self.
--	------------------

---

**Attributes:**


---

<code><a href="#">blobSize</a></code>
<code><a href="#">blobUri</a></code>
<code><a href="#">numFrames</a></code>
<code><a href="#">numNCEPerThread</a></code>
<code><a href="#">numThreads</a></code>

---

`\_\_init\_\_(*args, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

**property blobSize**

**property blobUri**

**property numFrames**

**property numNCEPerThread**

**property numThreads**

**class** [depthai.Node](#)

Bases: [pybind11\\_builtins.pybind11\\_object](#)

Abstract Node

**Classes:**

<i>Connection</i>	Connection between an Input and Output
<i>Id</i>	Node identificator.
<i>Input</i>	
<i>Output</i>	

**Methods:**

<i>__init__</i> (*args, **kwargs)	Initialize self.
<i>getAssets</i> (self)	Retrieves all nodes assets
<i>getInputs</i> (self)	Retrieves all nodes inputs
<i>getName</i> (self)	Retrieves nodes name
<i>getOutputs</i> (self)	Retrieves all nodes outputs

**Attributes:**

<i>id</i>	Id of node
-----------	------------

**class Connection**Bases: `pybind11_builtins.pybind11_object`

Connection between an Input and Output

**Methods:**

<i>__init__</i> (*args, **kwargs)	Initialize self.
-----------------------------------	------------------

**Attributes:**

<i>inputId</i>
<i>inputName</i>
<i>outputId</i>
<i>outputName</i>

*\_\_init\_\_*(\*args, \*\*kwargs)  
Initialize self. See `help(type(self))` for accurate signature.

**property inputId****property inputName****property outputId****property outputName****class Id**Bases: `pybind11_builtins.pybind11_object`

Node identificator. Unique for every node on a single Pipeline

**Methods:**

<i>__init__</i> (*args, **kwargs)	Initialize self.
-----------------------------------	------------------

*\_\_init\_\_*(\*args, \*\*kwargs)



Initialize self. See `help(type(self))` for accurate signature.

#### class Input

Bases: `pybind11_builtins.pybind11_object`

##### Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBlocking(self)</code>	Get input queue behavior
<code>getQueueSize(self)</code>	Get input queue size.
<code>setBlocking(self, blocking)</code>	Overrides default input queue behavior.
<code>setQueueSize(self, size)</code>	Overrides default input queue size.

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

**getBlocking** (*self*: `depthai.Node.Input`) → `bool`

Get input queue behavior

**Returns** True blocking, false overwriting

**getQueueSize** (*self*: `depthai.Node.Input`) → `int`

Get input queue size.

**Returns** Maximum input queue size

**setBlocking** (*self*: `depthai.Node.Input`, *blocking*: `bool`) → `None`

Overrides default input queue behavior.

**Parameter blocking:** True blocking, false overwriting

**setQueueSize** (*self*: `depthai.Node.Input`, *size*: `int`) → `None`

Overrides default input queue size. If queue size fills up, behavior depends on *blocking* attribute

**Parameter size:** Maximum input queue size

#### class Output

Bases: `pybind11_builtins.pybind11_object`

##### Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>canConnect(self, in)</code>	Check if connection is possible
<code>getConnections(self)</code>	Retrieve all connections from this output
<code>link(self, in)</code>	Link current output to input.
<code>unlink(self, in)</code>	Unlink a previously linked connection

`__init__(*args, **kwargs)`

Initialize self. See `help(type(self))` for accurate signature.

**canConnect** (*self*: `depthai.Node.Output`, *in*: `depthai.Node.Input`) → `bool`

Check if connection is possible

**Parameter in:** Input to connect to

**Returns** True if connection is possible, false otherwise

**getConnections** (*self*: `depthai.Node.Output`) → `List[dai::Node::Connection]`

Retrieve all connections from this output

**Returns** Vector of connections

**link** (*self*: `depthai.Node.Output`, *in*: `depthai.Node.Input`) → `None`

Link current output to input.

Throws an error if this output cannot be linked to given input, or if they are already linked

**Parameter in:** Input to link to

**unlink** (*self*: *depthai.Node.Output*, *in*: *depthai.Node.Input*) → *None*

Unlink a previously linked connection

Throws an error if not linked.

**Parameter in:** Input from which to unlink from

**\_\_init\_\_** (*\*args*, *\*\*kwargs*)

Initialize self. See help(type(self)) for accurate signature.

**getAssets** (*self*: *depthai.Node*) → List[*depthai.Asset*]

Retrieves all nodes assets

**getInputs** (*self*: *depthai.Node*) → List[dai::Node::Input]

Retrieves all nodes inputs

**getName** (*self*: *depthai.Node*) → str

Retrieves nodes name

**getOutputs** (*self*: *depthai.Node*) → List[dai::Node::Output]

Retrieves all nodes outputs

**property id**

Id of node

**class** *depthai.OpenVINO*

Bases: *pybind11\_builtins.pybind11\_object*

Support for basic OpenVINO related actions like version identification of neural network blobs,...

**Attributes:**

---

*VERSION\_2020\_1*

---

*VERSION\_2020\_2*

---

*VERSION\_2020\_3*

---

*VERSION\_2020\_4*

---

*VERSION\_2021\_1*

---

*VERSION\_2021\_2*

---

**Classes:**

---

*Version*

---

OpenVINO Version supported version information

---

**Methods:**

---

*\_\_init\_\_* (*\*args*, *\*\*kwargs*)

---

Initialize self.

---

*areVersionsBlobCompatible* (*v1*, *v2*)

---

Checks whether two blob versions are compatible

---

*getBlobLatestSupportedVersion* (*majorVersion*, *minorVersion*, *...)*

---

Returns latest potentially supported version by a given blob version.

---

*getBlobSupportedVersions* (*majorVersion*, *minorVersion*, *...)*

---

Returns a list of potentially supported versions for a specified blob major and minor versions.

---

*getVersionName* (*version*)

---

Returns string representation of a given version

---

*getVersions* ()

---

**returns** Supported versions

---

continues on next page

Table 107 – continued from previous page

<code>parseVersionName(versionString)</code>	Creates Version from string representation.
<pre> <b>VERSION_2020_1</b> = &lt;Version.VERSION_2020_1: 0&gt; <b>VERSION_2020_2</b> = &lt;Version.VERSION_2020_2: 1&gt; <b>VERSION_2020_3</b> = &lt;Version.VERSION_2020_3: 2&gt; <b>VERSION_2020_4</b> = &lt;Version.VERSION_2020_4: 3&gt; <b>VERSION_2021_1</b> = &lt;Version.VERSION_2021_1: 4&gt; <b>VERSION_2021_2</b> = &lt;Version.VERSION_2021_2: 5&gt;  <b>class Version</b>     Bases: pybind11_builtins.pybind11_object     OpenVINO Version supported version information      Members:         VERSION_2020_1         VERSION_2020_2         VERSION_2020_3         VERSION_2020_4         VERSION_2021_1         VERSION_2021_2      Attributes: </pre>	
<code>VERSION_2020_1</code>	
<code>VERSION_2020_2</code>	
<code>VERSION_2020_3</code>	
<code>VERSION_2020_4</code>	
<code>VERSION_2021_1</code>	
<code>VERSION_2021_2</code>	
<code>name</code>	
<code>value</code>	
<b>Methods:</b>	
<code>__init__(self, value)</code>	
<pre> <b>VERSION_2020_1</b> = &lt;Version.VERSION_2020_1: 0&gt; <b>VERSION_2020_2</b> = &lt;Version.VERSION_2020_2: 1&gt; <b>VERSION_2020_3</b> = &lt;Version.VERSION_2020_3: 2&gt; <b>VERSION_2020_4</b> = &lt;Version.VERSION_2020_4: 3&gt; <b>VERSION_2021_1</b> = &lt;Version.VERSION_2021_1: 4&gt; <b>VERSION_2021_2</b> = &lt;Version.VERSION_2021_2: 5&gt;  __init__(self: depthai.OpenVINO.Version, value: int) → None </pre>	

**property name**

**property value**

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**static areVersionsBlobCompatible** (v1: *dai::OpenVINO::Version*, v2: *dai::OpenVINO::Version*) → **bool**

Checks whether two blob versions are compatible

**OpenVINO.getBlobLatestSupportedVersion** (majorVersion: **int**, majorVersion: **int**) → **dai::OpenVINO::Version**

Returns latest potentially supported version by a given blob version.

**Parameter majorVersion:** Major version from OpenVINO blob

**Parameter minorVersion:** Minor version from OpenVINO blob

**Returns** Latest potentially supported version

**OpenVINO.getBlobSupportedVersions** (majorVersion: **int**, majorVersion: **int**) → **List[dai::OpenVINO::Version]**

Returns a list of potentially supported versions for a specified blob major and minor versions.

**Parameter majorVersion:** Major version from OpenVINO blob

**Parameter minorVersion:** Minor version from OpenVINO blob

**Returns** Vector of potentially supported versions

**static getVersionName** (version: *dai::OpenVINO::Version*) → **str**

Returns string representation of a given version

**Parameter version:** OpenVINO version

**Returns** Name of a given version

**static getVersions** () → **List[dai::OpenVINO::Version]**

**Returns** Supported versions

**static parseVersionName** (versionString: *str*) → *dai::OpenVINO::Version*

Creates Version from string representation. Throws if not possible.

**Parameter versionString:** Version as string

**Returns** Version object if successful

**class depthai.Pipeline**

Bases: `pybind11_builtins.pybind11_object`

Represents the pipeline, set of nodes and connections between them

**Methods:**

<code>__init__(self)</code>	Constructs a new pipeline
<code>createColorCamera(self)</code>	
<code>createImageManip(self)</code>	
<code>createMobileNetDetectionNetwork(self)</code>	
<code>createMobileNetSpatialDetectionNetwork(self)</code>	
<code>createMonoCamera(self)</code>	

continues on next page

Table 110 – continued from previous page

<code>createNeuralNetwork(self)</code>	
<code>createSPIOut(self)</code>	
<code>createSpatialLocationCalculator(self)</code>	
<code>createStereoDepth(self)</code>	
<code>createSystemLogger(self)</code>	
<code>createVideoEncoder(self)</code>	
<code>createXLinkIn(self)</code>	
<code>createXLinkOut(self)</code>	
<code>createYoloDetectionNetwork(self)</code>	
<code>createYoloSpatialDetectionNetwork(self)</code>	
<code>getAllAssets(self)</code>	Get assets on the pipeline includes nodes assets
<code>getAllNodes(*args, **kwargs)</code>	Overloaded function.
<code>getAssetManager(*args, **kwargs)</code>	Overloaded function.
<code>getConnectionMap(self)</code>	Get a reference to internal connection representation
<code>getConnections(self)</code>	Get all connections
<code>getGlobalProperties(self)</code>	<b>returns</b> Global properties of current pipeline
<code>getNode(*args, **kwargs)</code>	Overloaded function.
<code>getNodeMap(self)</code>	Get a reference to internal node map
<code>link(self, arg0, arg1)</code>	Link output to an input.
<code>remove(self, node)</code>	Removes a node from pipeline
<code>setOpenVINOVersion(self, version)</code>	Set a specific OpenVINO version to use with this pipeline
<code>unlink(self, arg0, arg1)</code>	Unlink output from an input.

`__init__ (self: depthai.Pipeline) → None`

Constructs a new pipeline

`createColorCamera (self: depthai.Pipeline) → depthai.ColorCamera`

`createImageManip (self: depthai.Pipeline) → depthai.ImageManip`

`createMobileNetDetectionNetwork (self: depthai.Pipeline) → depthai.MobileNetDetectionNetwork`

`createMobileNetSpatialDetectionNetwork (self: depthai.Pipeline) → depthai.MobileNetSpatialDetectionNetwork`

`createMonoCamera (self: depthai.Pipeline) → depthai.MonoCamera`

`createNeuralNetwork (self: depthai.Pipeline) → depthai.NeuralNetwork`

`createSPIOut (self: depthai.Pipeline) → depthai.SPIOut`

`createSpatialLocationCalculator (self: depthai.Pipeline) → depthai.SpatialLocationCalculator`

`createStereoDepth (self: depthai.Pipeline) → depthai.StereoDepth`

`createSystemLogger (self: depthai.Pipeline) → depthai.SystemLogger`

`createVideoEncoder (self: depthai.Pipeline) → depthai.VideoEncoder`

`createXLinkIn (self: depthai.Pipeline) → depthai.XLinkIn`

`createXLinkOut (self: depthai.Pipeline) → depthai.XLinkOut`

`createYoloDetectionNetwork (self: depthai.Pipeline) → depthai.YoloDetectionNetwork`

**createYoloSpatialDetectionNetwork** (*self*: [depthai.Pipeline](#)) → [depthai.YoloSpatialDetectionNetwork](#)

**getAllAssets** (*self*: [depthai.Pipeline](#)) → [depthai.AssetManager](#)  
Get assets on the pipeline includes nodes assets

**getAllNodes** (*\*args, \*\*kwargs*)  
Overloaded function.

1. `getAllNodes(self: depthai.Pipeline) -> List[depthai.Node]`  
Get a vector of all nodes
2. `getAllNodes(self: depthai.Pipeline) -> List[depthai.Node]`  
Get a vector of all nodes

**getAssetManager** (*\*args, \*\*kwargs*)  
Overloaded function.

1. `getAssetManager(self: depthai.Pipeline) -> depthai.AssetManager`  
Get pipelines AssetManager as reference
2. `getAssetManager(self: depthai.Pipeline) -> depthai.AssetManager`  
Get pipelines AssetManager as reference

**getConnectionMap** (*self*: [depthai.Pipeline](#)) → Dict[int, Set[[depthai.Node.Connection](#)]]  
Get a reference to internal connection representation

**getConnections** (*self*: [depthai.Pipeline](#)) → List[[depthai.Node.Connection](#)]  
Get all connections

**getGlobalProperties** (*self*: [depthai.Pipeline](#)) → [depthai.GlobalProperties](#)  
**Returns** Global properties of current pipeline

**getNode** (*\*args, \*\*kwargs*)  
Overloaded function.

1. `getNode(self: depthai.Pipeline, arg0: int) -> depthai.Node`  
Get node with id if it exists, nullptr otherwise
2. `getNode(self: depthai.Pipeline, arg0: int) -> depthai.Node`  
Get node with id if it exists, nullptr otherwise

**getNodeMap** (*self*: [depthai.Pipeline](#)) → Dict[int, [depthai.Node](#)]  
Get a reference to internal node map

**link** (*self*: [depthai.Pipeline](#), *arg0*: [depthai.Node.Output](#), *arg1*: [depthai.Node.Input](#)) → None  
Link output to an input. Both nodes must be on the same pipeline  
Throws an error if they aren't or cannot be connected  
**Parameter out:** Nodes output to connect from  
**Parameter in:** Nodes input to connect to

**remove** (*self*: [depthai.Pipeline](#), *node*: [depthai.Node](#)) → None  
Removes a node from pipeline

**setOpenVINOVersion** (*self*: [depthai.Pipeline](#), *version*: [depthai.OpenVINO.Version](#) = `<Version.VERSION_2020_1: 0>`) → None  
Set a specific OpenVINO version to use with this pipeline

**unlink** (*self*: depthai.Pipeline, *arg0*: depthai.Node.Output, *arg1*: depthai.Node.Input) → None

Unlink output from an input.

Throws an error if link doesn't exists

**Parameter out:** Nodes output to unlink from

**Parameter in:** Nodes input to unlink to

**class** depthai.Point2f

Bases: pybind11\_builtins.pybind11\_object

Point2f structure

x and y coordinates that define a 2D point.

**Methods:**

---

<code>__init__</code> (*args, **kwargs)	Overloaded function.
---	----------------------

---

**Attributes:**

---

<code>x</code>
<code>y</code>

---

`__init__`(\*args, \*\*kwargs)

Overloaded function.

1. `__init__(self: depthai.Point2f) -> None`
2. `__init__(self: depthai.Point2f, arg0: float, arg1: float) -> None`

**property x**

**property y**

**class** depthai.Point3f

Bases: pybind11\_builtins.pybind11\_object

Point3f structure

x,y,z coordinates that define a 3D point.

**Methods:**

---

<code>__init__</code> (*args, **kwargs)	Overloaded function.
---	----------------------

---

**Attributes:**

---

<code>x</code>
<code>y</code>
<code>z</code>

---

`__init__`(\*args, \*\*kwargs)

Overloaded function.

1. `__init__(self: depthai.Point3f) -> None`
2. `__init__(self: depthai.Point3f, arg0: float, arg1: float, arg2: float) -> None`

**property x**

**property** *y*

**property** *z*

**class** `depthai.RawBuffer`

Bases: `pybind11_builtins.pybind11_object`

**Methods:**

---

`__init__`(self)

---

**Attributes:**

---

*data*

---

`__init__`(self: `depthai.RawBuffer`) → `None`

**property** *data*

**class** `depthai.RawCameraControl`

Bases: `depthai.RawBuffer`

**Classes:**

---

<i>AntiBandingMode</i>	Members:
------------------------	----------

---

<i>AutoFocusMode</i>	Members:
----------------------	----------

---

<i>AutoWhiteBalanceMode</i>	Members:
-----------------------------	----------

---

<i>EffectMode</i>	Members:
-------------------	----------

---

<i>SceneMode</i>	Members:
------------------	----------

---

**Methods:**

---

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

---

**Attributes:**

---

*autoFocusMode*

---

*cmdMask*

---

*lensPosition*

---

**class** `AntiBandingMode`

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

MAINS\_50\_HZ

MAINS\_60\_HZ

AUTO

**Attributes:**



---

*AUTO*

---

*MAINS\_50\_HZ*

---

*MAINS\_60\_HZ*

---

*OFF*

---

*name*

---

*value*

---

**Methods:**


---

`__init__(self, value)`

---

**AUTO** = <AntiBandingMode.AUTO: 3>

**MAINS\_50\_HZ** = <AntiBandingMode.MAINS\_50\_HZ: 1>

**MAINS\_60\_HZ** = <AntiBandingMode.MAINS\_60\_HZ: 2>

**OFF** = <AntiBandingMode.OFF: 0>

`__init__(self: depthai.RawCameraControl.AntiBandingMode, value: int) → None`

**property name**

**property value**

**class AutoFocusMode**

Bases: `pybind11_builtins.pybind11_object`

Members:

OFF

AUTO

MACRO

CONTINUOUS\_VIDEO

CONTINUOUS\_PICTURE

EDOF

**Attributes:**


---

*AUTO*

---

*CONTINUOUS\_PICTURE*

---

*CONTINUOUS\_VIDEO*

---

*EDOF*

---

*MACRO*

---

*OFF*

---

*name*

---

*value*

---

**Methods:**


---

`__init__(self, value)`

---

**AUTO** = <AutoFocusMode.AUTO: 1>

```
CONTINUOUS_PICTURE = <AutoFocusMode.CONTINUOUS_PICTURE: 4>
CONTINUOUS_VIDEO = <AutoFocusMode.CONTINUOUS_VIDEO: 3>
EDOF = <AutoFocusMode.EDOF: 5>
MACRO = <AutoFocusMode.MACRO: 2>
OFF = <AutoFocusMode.OFF: 0>

__init__(self: depthai.RawCameraControl.AutoFocusMode, value: int) → None

property name
property value
```

```
class AutoWhiteBalanceMode
    Bases: pybind11_builtins.pybind11_object
```

Members:

```
    OFF
    AUTO
    INCANDESCENT
    FLUORESCENT
    WARM_FLUORESCENT
    DAYLIGHT
    CLOUDY_DAYLIGHT
    TWILIGHT
    SHADE
```

Attributes:

<i>AUTO</i>
<i>CLOUDY_DAYLIGHT</i>
<i>DAYLIGHT</i>
<i>FLUORESCENT</i>
<i>INCANDESCENT</i>
<i>OFF</i>
<i>SHADE</i>
<i>TWILIGHT</i>
<i>WARM_FLUORESCENT</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__(self, value)</i>
------------------------------

```
AUTO = <AutoWhiteBalanceMode.AUTO: 1>
CLOUDY_DAYLIGHT = <AutoWhiteBalanceMode.CLOUDY_DAYLIGHT: 6>
DAYLIGHT = <AutoWhiteBalanceMode.DAYLIGHT: 5>
FLUORESCENT = <AutoWhiteBalanceMode.FLUORESCENT: 3>
```

```

INCANDESCENT = <AutoWhiteBalanceMode.INCANDESCENT: 2>
OFF = <AutoWhiteBalanceMode.OFF: 0>
SHADE = <AutoWhiteBalanceMode.SHADE: 8>
TWILIGHT = <AutoWhiteBalanceMode.TWILIGHT: 7>
WARM_FLUORESCENT = <AutoWhiteBalanceMode.WARM_FLUORESCENT: 4>
__init__(self: depthai.RawCameraControl.AutoWhiteBalanceMode, value: int) → None
property name
property value

```

```

class EffectMode
Bases: pybind11_builtins.pybind11_object

```

Members:

```

OFF
MONO
NEGATIVE
SOLARIZE
SEPIA
POSTERIZE
WHITEBOARD
BLACKBOARD
AQUA

```

Attributes:

<i>AQUA</i>
<i>BLACKBOARD</i>
<i>MONO</i>
<i>NEGATIVE</i>
<i>OFF</i>
<i>POSTERIZE</i>
<i>SEPIA</i>
<i>SOLARIZE</i>
<i>WHITEBOARD</i>
<i>name</i>
<i>value</i>

Methods:

<i>__init__(self, value)</i>
------------------------------

```

AQUA = <EffectMode.AQUA: 8>
BLACKBOARD = <EffectMode.BLACKBOARD: 7>
MONO = <EffectMode.MONO: 1>
NEGATIVE = <EffectMode.NEGATIVE: 2>

```

OFF = <EffectMode.OFF: 0>

POSTERIZE = <EffectMode.POSTERIZE: 5>

SEPIA = <EffectMode.SEPIA: 4>

SOLARIZE = <EffectMode.SOLARIZE: 3>

WHITEBOARD = <EffectMode.WHITEBOARD: 6>

`__init__(self: depthai.RawCameraControl.EffectMode, value: int) → None`

property name

property value

**class SceneMode**

Bases: `pybind11_builtins.pybind11_object`

Members:

UNSUPPORTED

FACE\_PRIORITY

ACTION

PORTRAIT

LANDSCAPE

NIGHT

NIGHT\_PORTRAIT

THEATRE

BEACH

SNOW

SUNSET

STEADYPHOTO

FIREWORKS

SPORTS

PARTY

CANDLELIGHT

BARCODE

**Attributes:**

---

*ACTION*

---

*BARCODE*

---

*BEACH*

---

*CANDLELIGHT*

---

*FACE\_PRIORITY*

---

*FIREWORKS*

---

*LANDSCAPE*

---

*NIGHT*

---

*NIGHT\_PORTRAIT*

---

continues on next page

Table 128 – continued from previous page

<i>PARTY</i>
<i>PORTRAIT</i>
<i>SNOW</i>
<i>SPORTS</i>
<i>STEADYPHOTO</i>
<i>SUNSET</i>
<i>THEATRE</i>
<i>UNSUPPORTED</i>
<i>name</i>
<i>value</i>

**Methods:**

<code>__init__(self, value)</code>
<code>ACTION = &lt;SceneMode.ACTION: 2&gt;</code>
<code>BARCODE = &lt;SceneMode.BARCODE: 16&gt;</code>
<code>BEACH = &lt;SceneMode.BEACH: 8&gt;</code>
<code>CANDLELIGHT = &lt;SceneMode.CANDLELIGHT: 15&gt;</code>
<code>FACE_PRIORITY = &lt;SceneMode.FACE_PRIORITY: 1&gt;</code>
<code>FIREWORKS = &lt;SceneMode.FIREWORKS: 12&gt;</code>
<code>LANDSCAPE = &lt;SceneMode.LANDSCAPE: 4&gt;</code>
<code>NIGHT = &lt;SceneMode.NIGHT: 5&gt;</code>
<code>NIGHT_PORTRAIT = &lt;SceneMode.NIGHT_PORTRAIT: 6&gt;</code>
<code>PARTY = &lt;SceneMode.PARTY: 14&gt;</code>
<code>PORTRAIT = &lt;SceneMode.PORTRAIT: 3&gt;</code>
<code>SNOW = &lt;SceneMode.SNOW: 9&gt;</code>
<code>SPORTS = &lt;SceneMode.SPORTS: 13&gt;</code>
<code>STEADYPHOTO = &lt;SceneMode.STEADYPHOTO: 11&gt;</code>
<code>SUNSET = &lt;SceneMode.SUNSET: 10&gt;</code>
<code>THEATRE = &lt;SceneMode.THEATRE: 7&gt;</code>
<code>UNSUPPORTED = &lt;SceneMode.UNSUPPORTED: 0&gt;</code>
<code>__init__(self: depthai.RawCameraControl.SceneMode, value: int) → None</code>
<b>property name</b>
<b>property value</b>
<code>__init__(*args, **kwargs)</code>
Initialize self. See help(type(self)) for accurate signature.
<b>property autoFocusMode</b>
<b>property cmdMask</b>
<b>property lensPosition</b>

**class** `depthai.RawImageManipConfig`

Bases: `depthai.RawBuffer`

**Classes:**

---

`CropConfig`

---

`CropRect`

---

`FormatConfig`

---

`ResizeConfig`

---

**Methods:**

---

`__init__`(self)

---

**Attributes:**

---

`cropConfig`

---

`enableCrop`

---

`enableFormat`

---

`enableResize`

---

`formatConfig`

---

`resizeConfig`

---

**class** `CropConfig`

Bases: `pybind11_builtins.pybind11_object`

**Methods:**

---

`__init__`(self)

---

**Attributes:**

---

`cropRatio`

---

`cropRect`

---

`cropRotatedRect`

---

`enableCenterCropRectangle`

---

`enableRotatedRect`

---

`normalizedCoords`

---

`widthHeightAspectRatio`

---

`__init__`(self: `depthai.RawImageManipConfig.CropConfig`) → None

**property** `cropRatio`

**property** `cropRect`

**property** `cropRotatedRect`

**property** `enableCenterCropRectangle`

**property** `enableRotatedRect`

**property** `normalizedCoords`

**property** `widthHeightAspectRatio`

**class CropRect**Bases: `pybind11_builtins.pybind11_object`**Methods:**

---

`__init__(self)`

---

**Attributes:**

---

`xmax`

---

---

`xmin`

---

---

`ymax`

---

---

`ymin`

---

`__init__(self: depthai.RawImageManipConfig.CropRect) → None`**property xmax****property xmin****property ymax****property ymin****class FormatConfig**Bases: `pybind11_builtins.pybind11_object`**Methods:**

---

`__init__(self)`

---

**Attributes:**

---

`flipHorizontal`

---

---

`type`

---

`__init__(self: depthai.RawImageManipConfig.FormatConfig) → None`**property flipHorizontal****property type****class ResizeConfig**Bases: `pybind11_builtins.pybind11_object`**Methods:**

---

`__init__(self)`

---

**Attributes:**

---

`bgBlue`

---

---

`bgGreen`

---

---

`bgRed`

---

---

`enableRotation`

---

continues on next page

Table 140 – continued from previous page

<i>enableWarp4pt</i>
<i>enableWarpMatrix</i>
<i>height</i>
<i>keepAspectRatio</i>
<i>lockAspectRatioFill</i>
<i>normalizedCoords</i>
<i>rotationAngleDeg</i>
<i>warpBorderReplicate</i>
<i>warpFourPoints</i>
<i>warpMatrix3x3</i>
<i>width</i>

```
__init__(self: depthai.RawImageManipConfig.ResizeConfig) → None
property bgBlue
property bgGreen
property bgRed
property enableRotation
property enableWarp4pt
property enableWarpMatrix
property height
property keepAspectRatio
property lockAspectRatioFill
property normalizedCoords
property rotationAngleDeg
property warpBorderReplicate
property warpFourPoints
property warpMatrix3x3
property width
__init__(self: depthai.RawImageManipConfig) → None
property cropConfig
property enableCrop
property enableFormat
property enableResize
property formatConfig
property resizeConfig
class depthai.RawImgDetections
  Bases: depthai.RawBuffer
  Methods:
```



---

`__init__(self)`


---

**Attributes:**


---

`detections`


---

`__init__(self: depthai.RawImgDetections) → None`
**property** `detections`**class** `depthai.RawImgFrame`Bases: `depthai.RawBuffer`**Classes:**


---

`Specs`


---

`Type`

Members:

**Methods:**


---

`__init__(self)`


---

**Attributes:**


---

`category`


---

`fb`


---

`instanceNum`


---



---

`sequenceNum`


---

`ts`**class** `Specs`Bases: `pybind11_builtins.pybind11_object`**Methods:**


---

`__init__(*args, **kwargs)`


---

Initialize self.

**Attributes:**


---

`bytesPP`


---

`height`


---

`p1Offset`


---

`p2Offset`


---

`p3Offset`


---

`stride`


---

`type`


---

`width`


---

`__init__(*args, **kwargs)`


---

Initialize self. See `help(type(self))` for accurate signature.**property** `bytesPP`

```
    property height
    property p1Offset
    property p2Offset
    property p3Offset
    property stride
    property type
    property width

class Type
    Bases: pybind11_builtins.pybind11_object
    Members:
    YUV422i
    YUV444p
    YUV420p
    YUV422p
    YUV400p
    RGBA8888
    RGB161616
    RGB888p
    BGR888p
    RGB888i
    BGR888i
    RGBF16F16F16p
    BGRF16F16F16p
    RGBF16F16F16i
    BGRF16F16F16i
    GRAY8
    GRAYF16
    LUT2
    LUT4
    LUT16
    RAW16
    RAW14
    RAW12
    RAW10
    RAW8
    PACK10
```

PACK12

YUV444i

NV12

NV21

BITSTREAM

HDR

NONE

**Attributes:**

<i>BGR888i</i>
<i>BGR888p</i>
<i>BGRF16F16F16i</i>
<i>BGRF16F16F16p</i>
<i>BITSTREAM</i>
<i>GRAY8</i>
<i>GRAYF16</i>
<i>HDR</i>
<i>LUT16</i>
<i>LUT2</i>
<i>LUT4</i>
<i>NONE</i>
<i>NV12</i>
<i>NV21</i>
<i>PACK10</i>
<i>PACK12</i>
<i>RAW10</i>
<i>RAW12</i>
<i>RAW14</i>
<i>RAW16</i>
<i>RAW8</i>
<i>RGB161616</i>
<i>RGB888i</i>
<i>RGB888p</i>
<i>RGBA8888</i>
<i>RGBF16F16F16i</i>
<i>RGBF16F16F16p</i>
<i>YUV400p</i>
<i>YUV420p</i>
<i>YUV422i</i>
<i>YUV422p</i>
<i>YUV444i</i>
<i>YUV444p</i>
<i>name</i>
<i>value</i>

**Methods:**

---

`__init__(self, value)`

---

```
BGR888i = <Type.BGR888i: 10>
BGR888p = <Type.BGR888p: 8>
BGRF16F16F16i = <Type.BGRF16F16F16i: 14>
BGRF16F16F16p = <Type.BGRF16F16F16p: 12>
BITSTREAM = <Type.BITSTREAM: 30>
GRAY8 = <Type.GRAY8: 15>
GRAYF16 = <Type.GRAYF16: 16>
HDR = <Type.HDR: 31>
LUT16 = <Type.LUT16: 19>
LUT2 = <Type.LUT2: 17>
LUT4 = <Type.LUT4: 18>
NONE = <Type.NONE: 32>
NV12 = <Type.NV12: 28>
NV21 = <Type.NV21: 29>
PACK10 = <Type.PACK10: 25>
PACK12 = <Type.PACK12: 26>
RAW10 = <Type.RAW10: 23>
RAW12 = <Type.RAW12: 22>
RAW14 = <Type.RAW14: 21>
RAW16 = <Type.RAW16: 20>
RAW8 = <Type.RAW8: 24>
RGB161616 = <Type.RGB161616: 6>
RGB888i = <Type.RGB888i: 9>
RGB888p = <Type.RGB888p: 7>
RGBA8888 = <Type.RGBA8888: 5>
RGBF16F16F16i = <Type.RGBF16F16F16i: 13>
RGBF16F16F16p = <Type.RGBF16F16F16p: 11>
YUV400p = <Type.YUV400p: 4>
YUV420p = <Type.YUV420p: 2>
YUV422i = <Type.YUV422i: 0>
YUV422p = <Type.YUV422p: 3>
YUV444i = <Type.YUV444i: 27>
YUV444p = <Type.YUV444p: 1>
__init__(self: depthai.RawImgFrame.Type, value: int) → None
```

**property** name  
**property** value  
`__init__` (*self*: `depthai.RawImgFrame`) → `None`  
**property** category  
**property** fb  
**property** instanceNum  
**property** sequenceNum  
**property** ts

**class** `depthai.RawNNData`  
Bases: `depthai.RawBuffer`

**Methods:**

---

`__init__`(*self*)

---

**Attributes:**

---

`batchSize`

---

`tensors`

---

`__init__` (*self*: `depthai.RawNNData`) → `None`

**property** batchSize  
**property** tensors

**class** `depthai.RawSpatialImgDetections`  
Bases: `depthai.RawBuffer`

**Methods:**

---

`__init__`(*self*)

---

**Attributes:**

---

`detections`

---

`__init__` (*self*: `depthai.RawSpatialImgDetections`) → `None`

**property** detections

**class** `depthai.RawSystemInformation`  
Bases: `depthai.RawBuffer`

System information of device  
Memory usage, cpu usage and chip temperature

**Methods:**

---

`__init__`(*self*)

---

**Attributes:**


---

*chipTemperature*


---

*cmxMemoryUsage*


---

*ddrMemoryUsage*


---

*leonCssCpuUsage*


---

*leonCssMemoryUsage*


---

*leonMssCpuUsage*


---

*leonMssMemoryUsage*


---

`__init__ (self: depthai.RawSystemInformation) → None`

**property** `chipTemperature`

**property** `cmxMemoryUsage`

**property** `ddrMemoryUsage`

**property** `leonCssCpuUsage`

**property** `leonCssMemoryUsage`

**property** `leonMssCpuUsage`

**property** `leonMssMemoryUsage`

**class** `depthai.Rect`

Bases: `pybind11_builtins.pybind11_object`

Rect structure

x,y coordinates together with width and height that define a rectangle. Can be either normalized [0,1] or absolute representation.

**Methods:**

<code>__init__ (*args, **kwargs)</code>	Overloaded function.
<code>area(self)</code>	Area (width*height) of the rectangle
<code>bottomRight(self)</code>	The bottom-right corner
<code>contains(self, arg0)</code>	Checks whether the rectangle contains the point.
<code>denormalize(self, width, height)</code>	Denormalize rectangle.
<code>empty(self)</code>	True if rectangle is empty.
<code>isNormalized(self)</code>	Whether rectangle is normalized (coordinates in [0,1] range) or not.
<code>normalize(self, width, height)</code>	Normalize rectangle.
<code>size(self)</code>	Size (width, height) of the rectangle
<code>topLeft(self)</code>	The top-left corner.

**Attributes:**


---

*height*


---

*width*


---

*x*


---

*y*


---

`__init__ (*args, **kwargs)`  
Overloaded function.

1. `__init__(self: depthai.Rect) -> None`
2. `__init__(self: depthai.Rect, arg0: float, arg1: float, arg2: float, arg3: float) -> None`
3. `__init__(self: depthai.Rect, arg0: depthai.Point2f, arg1: depthai.Point2f) -> None`
4. `__init__(self: depthai.Rect, arg0: depthai.Point2f, arg1: depthai.Size2f) -> None`

**area** (*self*: `depthai.Rect`)  $\rightarrow$  `float`

Area (width\*height) of the rectangle

**bottomRight** (*self*: `depthai.Rect`)  $\rightarrow$  `depthai.Point2f`

The bottom-right corner

**contains** (*self*: `depthai.Rect`, *arg0*: `depthai.Point2f`)  $\rightarrow$  `bool`

Checks whether the rectangle contains the point.

**denormalize** (*self*: `depthai.Rect`, *width*: `int`, *height*: `int`)  $\rightarrow$  `depthai.Rect`

Denormalize rectangle.

**Parameter width:** Destination frame width.

**Parameter height:** Destination frame height.

**empty** (*self*: `depthai.Rect`)  $\rightarrow$  `bool`

True if rectangle is empty.

**property height**

**isNormalized** (*self*: `depthai.Rect`)  $\rightarrow$  `bool`

Whether rectangle is normalized (coordinates in [0,1] range) or not.

**normalize** (*self*: `depthai.Rect`, *width*: `int`, *height*: `int`)  $\rightarrow$  `depthai.Rect`

Normalize rectangle.

**Parameter width:** Source frame width.

**Parameter height:** Source frame height.

**size** (*self*: `depthai.Rect`)  $\rightarrow$  `depthai.Size2f`

Size (width, height) of the rectangle

**topLeft** (*self*: `depthai.Rect`)  $\rightarrow$  `depthai.Point2f`

The top-left corner.

**property width**

**property x**

**property y**

**class** `depthai.RotatedRect`

Bases: `pybind11_builtins.pybind11_object`

**Methods:**

---

`__init__(self)`

---

**Attributes:**

---

`angle`

---

`center`

---

`size`

---

`__init__ (self: depthai.RotatedRect) → None`

**property angle**

**property center**

**property size**

**class** `depthai.SPIOut`

Bases: `depthai.Node`

SPIOut node. Sends messages over SPI.

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setBusId(self, id)</code>	Specifies SPI Bus number to use
<code>setStreamName(self, name)</code>	Specifies stream name over which the node will send data

**Attributes:**

<code>input</code>	Input for any type of messages to be transfered over SPI stream
--------------------	---

`__init__ (*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**property input**

Input for any type of messages to be transfered over SPI stream

Default queue is blocking with size 8

**setBusId** (`self: depthai.SPIOut, id: int`) → None

Specifies SPI Bus number to use

**Parameter id:** SPI Bus id

**setStreamName** (`self: depthai.SPIOut, name: str`) → None

Specifies stream name over which the node will send data

**Parameter name:** Stream name

**class** `depthai.Size2f`

Bases: `pybind11_builtins.pybind11_object`

**Methods:**

<code>__init__(*args, **kwargs)</code>	Overloaded function.
--	----------------------

**Attributes:**

<code>height</code>	
<code>width</code>	

`__init__ (*args, **kwargs)`

Overloaded function.

1. `__init__(self: depthai.Size2f) -> None`



2. `__init__(self: depthai.Size2f, arg0: float, arg1: float) -> None`

**property height**

**property width**

**class** `depthai.SpatialDetectionNetwork`

Bases: `depthai.DetectionNetwork`

SpatialDetectionNetwork node. Runs a neural inference on input image and calculates spatial location data.

**Classes:**

<i>Properties</i>	alias of <i>depthai.SpatialDetectionNetworkProperties</i>
-------------------	---

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setBoundingBoxScaleFactor(self, scaleFactor)</code>	Specifies scale factor for detected bounding boxes.
<code>setDepthLowerThreshold(self, lowerThreshold)</code>	Specifies lower threshold in millimeters for depth values which will be used to calculate spatial data
<code>setDepthUpperThreshold(self, upperThreshold)</code>	Specifies upper threshold in millimeters for depth values which will be used to calculate spatial data

**Attributes:**

<i>boundingBoxMapping</i>	Outputs mapping of detected bounding boxes relative to depth map
<i>input</i>	Input message with data to be inferred upon. Default queue is blocking with size 5
<i>inputDepth</i>	Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4
<i>out</i>	Outputs <code>ImgDetections</code> message that carries parsed detection results.
<i>passthrough</i>	Passthrough message on which the inference was performed.
<i>passthroughDepth</i>	Passthrough message for depth frame on which the spatial location calculation was performed.

**Properties**

alias of `depthai.SpatialDetectionNetworkProperties`

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<code>depthThresholds</code>
<code>detectedBBScaleFactor</code>

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**property boundingBoxMapping**

Outputs mapping of detected bounding boxes relative to depth map

Suitable for when displaying remapped bounding boxes on depth frame

**property input**

Input message with data to be inferred upon Default queue is blocking with size 5

**property inputDepth**

Input message with depth data used to retrieve spatial information about detected object Default queue is non-blocking with size 4

**property out**

Outputs ImgDetections message that carries parsed detection results.

**property passthrough**

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

**property passthroughDepth**

Passthrough message for depth frame on which the spatial location calculation was performed.

Suitable for when input queue is set to non-blocking behavior.

**setBoundingBoxScaleFactor** (*self*: [depthai.SpatialDetectionNetwork](#), *scaleFactor*: *float*) → *None*

Specifies scale factor for detected bounding boxes.

**Parameter scaleFactor:** Scale factor must be in the interval (0,1].

**setDepthLowerThreshold** (*self*: [depthai.SpatialDetectionNetwork](#), *lowerThreshold*: *int*) → *None*

Specifies lower threshold in millimeters for depth values which will used to calculate spatial data

**Parameter lowerThreshold:** LowerThreshold must be in the interval [0,upperThreshold] and less than upperThreshold.

**setDepthUpperThreshold** (*self*: [depthai.SpatialDetectionNetwork](#), *upperThreshold*: *int*) → *None*

Specifies upper threshold in millimeters for depth values which will used to calculate spatial data

**Parameter upperThreshold:** UpperThreshold must be in the interval (lowerThreshold,65535].

**class depthai.SpatialDetectionNetworkProperties**

Bases: [depthai.DetectionNetworkProperties](#)

Properties for SpatialDetectionNetwork

**Methods:**

---

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

---

**Attributes:**

---

<code>depthThresholds</code>
<code>detectedBBScaleFactor</code>

---

`__init__` (\*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**property depthThresholds**

**property detectedBBScaleFactor**

**class** depthai.SpatialImgDetection

Bases: *depthai.ImgDetection*

Spatial image detection structure

Contains image detection results together with spatial location data.

**Methods:**

---

*\_\_init\_\_*(self)

---

**Attributes:**

---

*spatialCoordinates*

---

*\_\_init\_\_*(self: depthai.SpatialImgDetection) → None

**property spatialCoordinates**

**class** depthai.SpatialImgDetections

Bases: *depthai.Buffer*

SpatialImgDetections message. Carries detection results together with spatial location data

**Methods:**

---

*\_\_init\_\_*(self)

---

**Attributes:**

---

*detections*

---

*\_\_init\_\_*(self: depthai.SpatialImgDetections) → None

**property detections**

**class** depthai.SpatialLocationCalculator

Bases: *depthai.Node*

SpatialLocationCalculator node. Calculates spatial location data on a set of ROIs on depth map.

**Classes:**

---

<i>Properties</i>	alias of <i>depthai.SpatialLocationCalculatorProperties</i>
-------------------	---

---

**Methods:**

---

<i>__init__</i> (*args, **kwargs)	Initialize self.
<i>setWaitForConfigInput</i> (self, wait)	Specify whether or not wait until configuration message arrives to inputConfig Input.

---

**Attributes:**

<i>initialConfig</i>	Initial config to use when calculating spatial location data.
<i>inputConfig</i>	Input <code>SpatialLocationCalculatorConfig</code> message with ability to modify parameters in runtime.
<i>inputDepth</i>	Input message with depth data used to retrieve spatial information about detected object.
<i>out</i>	Outputs <code>SpatialLocationCalculatorData</code> message that carries spatial location results.
<i>passthroughDepth</i>	Passthrough message on which the calculation was performed.

**Properties**

alias of `depthai.SpatialLocationCalculatorProperties` **Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<code>inputConfigSync</code>
<code>roiConfig</code>

`__init__(*args, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

**property initialConfig**

Initial config to use when calculating spatial location data.

**property inputConfig**

Input `SpatialLocationCalculatorConfig` message with ability to modify parameters in runtime. Default queue is non-blocking with size 4.

**property inputDepth**

Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4.

**property out**

Outputs `SpatialLocationCalculatorData` message that carries spatial location results.

**property passthroughDepth**

Passthrough message on which the calculation was performed. Suitable for when input queue is set to non-blocking behavior.

**setWaitForConfigInput** (*self*: `depthai.SpatialLocationCalculator`, *wait*: `bool`) → `None`

Specify whether or not wait until configuration message arrives to `inputConfig` Input.

**Parameter wait:** True to wait for configuration message, false otherwise.

**class depthai.SpatialLocationCalculatorConfig**

Bases: `depthai.Buffer`

`SpatialLocationCalculatorConfig` message. Carries ROI (region of interest) and threshold for depth calculation

**Methods:**

<code>__init__(self)</code>
-----------------------------

continues on next page

Table 180 – continued from previous page

<code>addROI(self, ROI)</code>	Add a new ROI to configuration data.
<code>getConfigData(self)</code>	Retrieve configuration data for SpatialLocationCalculator
<code>setROIs(self, ROIs)</code>	Set a vector of ROIs as configuration data.

`__init__ (self: depthai.SpatialLocationCalculatorConfig) → None`

**addROI** (*self*: depthai.SpatialLocationCalculatorConfig, *ROI*: depthai.SpatialLocationCalculatorConfigData) → None  
Add a new ROI to configuration data.

**Parameter roi:** Configuration parameters for ROI (region of interest)

**getConfigData** (*self*: depthai.SpatialLocationCalculatorConfig) → List[depthai.SpatialLocationCalculatorConfigData]  
Retrieve configuration data for SpatialLocationCalculator

**Returns** Vector of configuration parameters for ROIs (region of interests)

**setROIs** (*self*: depthai.SpatialLocationCalculatorConfig, *ROIs*: List[depthai.SpatialLocationCalculatorConfigData]) → None  
Set a vector of ROIs as configuration data.

**Parameter ROIs:** Vector of configuration parameters for ROIs (region of interests)

**class** depthai.SpatialLocationCalculatorConfigData

Bases: pybind11\_builtins.pybind11\_object

**Methods:**

---

`__init__ (self)`

---

**Attributes:**

---

`depthThresholds`

---

`roi`

---

`__init__ (self: depthai.SpatialLocationCalculatorConfigData) → None`

**property** depthThresholds

**property** roi

**class** depthai.SpatialLocationCalculatorConfigThresholds

Bases: pybind11\_builtins.pybind11\_object

Spatial location configuration thresholds structure

Contains configuration data for lower and upper threshold in millimeters for ROI. Values outside of threshold range will be ignored when calculating spatial coordinates from depth map.

**Methods:**

---

`__init__ (self)`

---

**Attributes:**

---

*lowerThreshold*

---

*upperThreshold*

---

**\_\_init\_\_** (*self*: depthai.SpatialLocationCalculatorConfigThresholds) → None**property** lowerThreshold**property** upperThreshold**class** depthai.SpatialLocationCalculatorDataBases: *depthai.Buffer*

SpatialLocationCalculatorData message. Carries spatial information (X,Y,Z) and their configuration parameters

**Methods:**

---

**\_\_init\_\_** (*self*)

---

*getSpatialLocations* (*self*)Retrieve configuration data for SpatialLocationCalculatorData.

---

**\_\_init\_\_** (*self*: depthai.SpatialLocationCalculatorData) → None**getSpatialLocations** (*self*: depthai.SpatialLocationCalculatorData) → List[*depthai.SpatialLocations*]

Retrieve configuration data for SpatialLocationCalculatorData.

**Returns** Vector of spatial location data, carrying spatial information (X,Y,Z)**class** depthai.SpatialLocationCalculatorPropertiesBases: *pybind11\_builtins.pybind11\_object*

Specify SpatialLocationCalculator options

**Methods:**

---

**\_\_init\_\_** (\*args, \*\*kwargs)

---

Initialize self.

---

**Attributes:**

---

*inputConfigSync*

---

*roiConfig*

---

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**property** inputConfigSync**property** roiConfig**class** depthai.SpatialLocationsBases: *pybind11\_builtins.pybind11\_object*

Spatial location information structure

Contains configuration data, average depth for the calculated ROI on depth map. Together with spatial coordinates: x,y,z. Origin is the center of ROI. Units are in millimeters.

**Methods:**

---

`__init__(self)`


---

**Attributes:**`config``depthAverage``spatialCoordinates`


---

`__init__(self: depthai.SpatialLocations) → None`**property** `config`**property** `depthAverage`**property** `spatialCoordinates`**class** `depthai.StereoDepth`Bases: `depthai.Node`

StereoDepth node. Compute stereo disparity and depth from left-right image pair.

**Classes:***Properties*alias of `depthai.StereoDepthProperties`


---

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>loadCalibrationData(self, data)</code>	Specify calibration data as a vector of bytes
<code>loadCalibrationFile(self, path)</code>	Specify local filesystem path to the calibration file
<code>setConfidenceThreshold(self, confThr)</code>	Confidence threshold for disparity calculation
<code>setEmptyCalibration(self)</code>	Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g.
<code>setExtendedDisparity(self, enable)</code>	Disparity range increased from 96 to 192, combined from full resolution and downsampled images.
<code>setInputResolution(self, width, height)</code>	Specify input resolution size
<code>setLeftRightCheck(self, enable)</code>	Computes and combines disparities in both L-R and R-L directions, and combine them.
<code>setMedianFilter(self, median)</code>	Parameter median:
<code>setOutputDepth(self, enable)</code>	Enable outputting 'depth' stream (converted from disparity).
<code>setOutputRectified(self, enable)</code>	Enable outputting rectified frames.
<code>setRectifyEdgeFillColor(self, color)</code>	Fill color for missing data at frame edges
<code>setRectifyMirrorFrame(self, enable)</code>	Mirror rectified frames
<code>setSubpixel(self, enable)</code>	Computes disparity with sub-pixel interpolation (5 fractional bits).

---

**Attributes:**`depth`Outputs `ImgFrame` message that carries RAW16 encoded (0..65535) depth data in millimeters.

continues on next page

Table 192 – continued from previous page

<i>disparity</i>	Outputs ImgFrame message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.
<i>left</i>	Input for left ImgFrame of left-right pair
<i>rectifiedLeft</i>	Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.
<i>rectifiedRight</i>	Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.
<i>right</i>	Input for right ImgFrame of left-right pair
<i>syncedLeft</i>	Passthrough ImgFrame message from ‘left’ Input.
<i>syncedRight</i>	Passthrough ImgFrame message from ‘right’ Input.

**Properties**alias of *depthai.StereoDepthProperties* **Classes:**

MedianFilter	Median filter config for disparity post-processing
--------------	--

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<code>calibration</code>
<code>confidenceThreshold</code>
<code>enableExtendedDisparity</code>
<code>enableLeftRightCheck</code>
<code>enableOutputDepth</code>
<code>enableOutputRectified</code>
<code>enableSubpixel</code>
<code>height</code>
<code>median</code>
<code>rectifyEdgeFillColor</code>
<code>rectifyMirrorFrame</code>
<code>width</code>

`__init__(*args, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

**property depth**

Outputs ImgFrame message that carries RAW16 encoded (0..65535) depth data in millimeters.

**property disparity**

Outputs ImgFrame message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.

**property left**

Input for left ImgFrame of left-right pair

Default queue is non-blocking with size 8

**loadCalibrationData** (*self*: *depthai.StereoDepth*, *data*: *List[int]*) → *None*

Specify calibration data as a vector of bytes



**Parameter path:** Calibration data. If empty use EEPROM

**loadCalibrationFile** (*self*: `depthai.StereoDepth`, *path*: *str*) → `None`  
Specify local filesystem path to the calibration file

**Parameter path:** Path to calibration file. If empty use EEPROM

**property rectifiedLeft**

Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.

**property rectifiedRight**

Outputs ImgFrame message that carries RAW8 encoded (grayscale) rectified frame data.

**property right**

Input for right ImgFrame of left-right pair

Default queue is non-blocking with size 8

**setConfidenceThreshold** (*self*: `depthai.StereoDepth`, *confThr*: *int*) → `None`  
Confidence threshold for disparity calculation

**Parameter confThr:** Confidence threshold value 0..255

**setEmptyCalibration** (*self*: `depthai.StereoDepth`) → `None`

Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g. sourced from recordings on the host)

**setExtendedDisparity** (*self*: `depthai.StereoDepth`, *enable*: *bool*) → `None`

Disparity range increased from 96 to 192, combined from full resolution and downscaled images.

Suitable for short range objects

**setInputResolution** (*self*: `depthai.StereoDepth`, *width*: *int*, *height*: *int*) → `None`  
Specify input resolution size

Optional if MonoCamera exists, otherwise necessary

**setLeftRightCheck** (*self*: `depthai.StereoDepth`, *enable*: *bool*) → `None`

Computes and combines disparities in both L-R and R-L directions, and combine them.

For better occlusion handling

**setMedianFilter** (*self*: `depthai.StereoDepth`, *median*: `dai::StereoDepthProperties::MedianFilter`) → `None`

**Parameter median:** Set kernel size for disparity/depth median filtering, or disable

**setOutputDepth** (*self*: `depthai.StereoDepth`, *enable*: *bool*) → `None`

Enable outputting 'depth' stream (converted from disparity). In certain configurations, this will disable 'disparity' stream

**setOutputRectified** (*self*: `depthai.StereoDepth`, *enable*: *bool*) → `None`

Enable outputting rectified frames. Optimizes computation on device side when disabled

**setRectifyEdgeFillColor** (*self*: `depthai.StereoDepth`, *color*: *int*) → `None`

Fill color for missing data at frame edges

**Parameter color:** Grayscale 0..255, or -1 to replicate pixels

**setRectifyMirrorFrame** (*self*: `depthai.StereoDepth`, *enable*: *bool*) → `None`

Mirror rectified frames

**Parameter enable:** True for normal disparity/depth, otherwise mirrored

**setSubpixel** (*self*: `depthai.StereoDepth`, *enable*: *bool*) → `None`  
Computes disparity with sub-pixel interpolation (5 fractional bits).

Suitable for long range

**property syncedLeft**  
Passthrough `ImgFrame` message from ‘left’ Input.

**property syncedRight**  
Passthrough `ImgFrame` message from ‘right’ Input.

**class** `depthai.StereoDepthProperties`  
Bases: `pybind11_builtins.pybind11_object`

Specify `StereoDepth` options

Classes:

---

<code>MedianFilter</code>	Median filter config for disparity post-processing
---------------------------	--

---

Methods:

---

<code>__init__</code> (*args, **kwargs)	Initialize self.
---	------------------

---

Attributes:

---

<code>calibration</code>
<code>confidenceThreshold</code>
<code>enableExtendedDisparity</code>
<code>enableLeftRightCheck</code>
<code>enableOutputDepth</code>
<code>enableOutputRectified</code>
<code>enableSubpixel</code>
<code>height</code>
<code>median</code>
<code>rectifyEdgeFillColor</code>
<code>rectifyMirrorFrame</code>
<code>width</code>

---

**class** `MedianFilter`  
Bases: `pybind11_builtins.pybind11_object`

Median filter config for disparity post-processing

Members:

`MEDIAN_OFF`  
`KERNEL_3x3`  
`KERNEL_5x5`  
`KERNEL_7x7`

Attributes:

---

<code>KERNEL_3x3</code>
-------------------------

---

continues on next page

Table 199 – continued from previous page

<code>KERNEL_5x5</code>
<code>KERNEL_7x7</code>
<code>MEDIAN_OFF</code>
<code>name</code>
<code>value</code>

**Methods:**

<code>__init__(self, value)</code>
------------------------------------

`KERNEL_3x3 = <MedianFilter.KERNEL_3x3: 3>`

`KERNEL_5x5 = <MedianFilter.KERNEL_5x5: 5>`

`KERNEL_7x7 = <MedianFilter.KERNEL_7x7: 7>`

`MEDIAN_OFF = <MedianFilter.MEDIAN_OFF: 0>`

`__init__(self: depthai.StereoDepthProperties.MedianFilter, value: int) → None`

property name

property value

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

property calibration

property confidenceThreshold

property enableExtendedDisparity

property enableLeftRightCheck

property enableOutputDepth

property enableOutputRectified

property enableSubpixel

property height

property median

property rectifyEdgeFillColor

property rectifyMirrorFrame

property width

**class** `depthai.SystemInformation`

Bases: `depthai.Buffer`

SystemInformation message. Carries memory usage, cpu usage and chip temperatures.

**Methods:**

<code>__init__(self)</code>
-----------------------------

**Attributes:**

---

*chipTemperature*

---

*cmxMemoryUsage*

---

*ddrMemoryUsage*

---

*leonCssCpuUsage*

---

*leonCssMemoryUsage*

---

*leonMssCpuUsage*

---

*leonMssMemoryUsage*

---

**\_\_init\_\_** (*self*: depthai.SystemInformation) → None

**property** chipTemperature

**property** cmxMemoryUsage

**property** ddrMemoryUsage

**property** leonCssCpuUsage

**property** leonCssMemoryUsage

**property** leonMssCpuUsage

**property** leonMssMemoryUsage

**class** depthai.SystemLogger

Bases: *depthai.Node*

SystemLogger node. Send system information periodically.

**Methods:**

---

*\_\_init\_\_* (\*args, \*\*kwargs)

Initialize self.

---

*setRate* (self, hz)

Specify logging rate, at which messages will be sent to out output

---

**Attributes:**

---

*out*

Outputs SystemInformation message that carries various system information like memory and CPU usage, temperatures, ...

---

**\_\_init\_\_** (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**property** out

Outputs SystemInformation message that carries various system information like memory and CPU usage, temperatures, ...

**setRate** (*self*: depthai.SystemLogger, *hz*: float) → None

Specify logging rate, at which messages will be sent to out output

**Parameter hz:** Sending rate in hertz (messages per second)

**class** depthai.SystemLoggerProperties

Bases: pybind11\_builtins.pybind11\_object

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<code>rateHz</code>
---------------------

`__init__(*args, **kwargs)`  
Initialize self. See help(type(self)) for accurate signature.

**property rateHz****class depthai.TensorInfo**Bases: `pybind11_builtins.pybind11_object`**Classes:**

<code>DataType</code>	Members:
<code>StorageOrder</code>	Members:

**Methods:**

<code>__init__(self)</code>
-----------------------------

**Attributes:**

<code>dataType</code>
<code>dims</code>
<code>name</code>
<code>numDimensions</code>
<code>offset</code>
<code>order</code>
<code>strides</code>

**class DataType**Bases: `pybind11_builtins.pybind11_object`**Members:**

FP16

U8F

INT

FP32

I8

**Attributes:**

<code>FP16</code>
<code>FP32</code>
<code>I8</code>
<code>INT</code>

continues on next page

Table 210 – continued from previous page

<i>U8F</i>
<i>name</i>
<i>value</i>
<b>Methods:</b>
<code>__init__(self, value)</code>
<pre> FP16 = &lt;DataType.FP16: 0&gt; FP32 = &lt;DataType.FP32: 3&gt; I8 = &lt;DataType.I8: 4&gt; INT = &lt;DataType.INT: 2&gt; U8F = &lt;DataType.U8F: 1&gt; __init__(self: depthai.TensorInfo.DataType, value: int) → None property name property value class StorageOrder   Bases: pybind11_builtins.pybind11_object   Members:     NHWC     NHCW     NCHW     HWC     CHW     WHC     HCW     WCH     CWH     NC     CN     C     H     W     Attributes: </pre>
<i>C</i>
<i>CHW</i>
<i>CN</i>
<i>CWH</i>

continues on next page

Table 212 – continued from previous page

<i>H</i>
<i>HCW</i>
<i>HWC</i>
<i>NC</i>
<i>NCHW</i>
<i>NHCW</i>
<i>NHWC</i>
<i>W</i>
<i>WCH</i>
<i>WHC</i>
<i>name</i>
<i>value</i>

**Methods:**

<code>__init__(self, value)</code>
<code>C = &lt;StorageOrder.C: 3&gt;</code>
<code>CHW = &lt;StorageOrder.CHW: 801&gt;</code>
<code>CN = &lt;StorageOrder.CN: 52&gt;</code>
<code>CWH = &lt;StorageOrder.CWH: 786&gt;</code>
<code>H = &lt;StorageOrder.H: 2&gt;</code>
<code>HCW = &lt;StorageOrder.HCW: 561&gt;</code>
<code>HWC = &lt;StorageOrder.HWC: 531&gt;</code>
<code>NC = &lt;StorageOrder.NC: 67&gt;</code>
<code>NCHW = &lt;StorageOrder.NCHW: 17185&gt;</code>
<code>NHCW = &lt;StorageOrder.NHCW: 16945&gt;</code>
<code>NHWC = &lt;StorageOrder.NHWC: 16915&gt;</code>
<code>W = &lt;StorageOrder.W: 1&gt;</code>
<code>WCH = &lt;StorageOrder.WCH: 306&gt;</code>
<code>WHC = &lt;StorageOrder.WHC: 291&gt;</code>
<code>__init__(self: depthai.TensorInfo.StorageOrder, value: int) → None</code>
<code>property name</code>
<code>property value</code>
<code>__init__(self: depthai.TensorInfo) → None</code>
<code>property dataType</code>
<code>property dims</code>
<code>property name</code>
<code>property numDimensions</code>
<code>property offset</code>

**property order****property strides****class** depthai.Timestamp

Bases: pybind11\_builtins.pybind11\_object

**Methods:**

---

`__init__(self)`

---

**Attributes:**

---

`nsec`

---

`sec`

---

`__init__(self: depthai.Timestamp) → None`**property nsec****property sec****class** depthai.VideoEncoderBases: *depthai.Node*

VideoEncoder node. Encodes frames into MJPEG, H264 or H265.

**Classes:**

---

<i>Properties</i>	alias	of	<i>depthai.</i>
	<i>VideoEncoderProperties</i>		

---

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getBitrate(self)</code>	Get bitrate in bps
<code>getBitrateKbps(self)</code>	Get bitrate in kbps
<code>getFrameRate(self)</code>	Get frame rate
<code>getHeight(self)</code>	Get input height
<code>getKeyframeFrequency(self)</code>	Get keyframe frequency
<code>getNumBFrames(self)</code>	Get number of B frames
<code>getNumFramesPool(self)</code>	Get number of frames in pool
<code>getProfile(self)</code>	Get profile
<code>getQuality(self)</code>	Get quality
<code>getRateControlMode(self)</code>	Get rate control mode
<code>getSize(self)</code>	Get input size
<code>getWidth(self)</code>	Get input width
<code>setBitrate(self, bitrateKbps)</code>	Set output bitrate in bps.
<code>setBitrateKbps(self, bitrateKbps)</code>	Set output bitrate in kbps.
<code>setDefaultProfilePreset(*args, **kwargs)</code>	Overloaded function.
<code>setFrameRate(self, frameRate)</code>	Sets expected frame rate
<code>setKeyframeFrequency(self, freq)</code>	Set keyframe frequency.
<code>setNumBFrames(self, numBFrames)</code>	Set number of B frames to be inserted
<code>setNumFramesPool(self, frames)</code>	Set number of frames in pool

continues on next page



Table 217 – continued from previous page

<code>setProfile(self, width, height, profile)</code>	Set encoding profile
<code>setQuality(self, quality)</code>	Set quality
<code>setRateControlMode(self, mode)</code>	Set rate control mode

**Attributes:**

<code>bitstream</code>	Outputs ImgFrame message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.
<code>input</code>	Input for NV12 ImgFrame to be encoded Default queue is blocking with size set by 'setNumFrame-sPool' (4).

**Properties**alias of `depthai.VideoEncoderProperties` **Classes:**

Profile	Encoding profile, H264, H265 or MJPEG
RateControlMode	Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
--	------------------

**Attributes:**

<code>bitrate</code>
<code>height</code>
<code>keyframeFrequency</code>
<code>maxBitrate</code>
<code>numBframes</code>
<code>numFramesPool</code>
<code>profile</code>
<code>quality</code>
<code>rateCtrlMode</code>
<code>width</code>

`__init__ (*args, **kwargs)`  
Initialize self. See `help(type(self))` for accurate signature.

**property bitstream**

Outputs ImgFrame message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.

**getBitrate** (*self*: `depthai.VideoEncoder`) → int  
Get bitrate in bps

**getBitrateKbps** (*self*: `depthai.VideoEncoder`) → int  
Get bitrate in kbps

**getFrameRate** (*self*: `depthai.VideoEncoder`) → int  
Get frame rate

**getHeight** (*self*: depthai.VideoEncoder) → int

Get input height

**getKeyframeFrequency** (*self*: depthai.VideoEncoder) → int

Get keyframe frequency

**getNumBframes** (*self*: depthai.VideoEncoder) → int

Get number of B frames

**getNumFramesPool** (*self*: depthai.VideoEncoder) → int

Get number of frames in pool

**Returns** Number of pool frames

**getProfile** (*self*: depthai.VideoEncoder) → dai::VideoEncoderProperties::Profile

Get profile

**getQuality** (*self*: depthai.VideoEncoder) → int

Get quality

**getRateControlMode** (*self*: depthai.VideoEncoder) → dai::VideoEncoderProperties::RateControlMode

Get rate control mode

**getSize** (*self*: depthai.VideoEncoder) → Tuple[int, int]

Get input size

**getWidth** (*self*: depthai.VideoEncoder) → int

Get input width

**property input**

Input for NV12 ImgFrame to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

**setBitrate** (*self*: depthai.VideoEncoder, *bitrateKbps*: int) → None

Set output bitrate in bps. Final bitrate depends on rate control mode

**setBitrateKbps** (*self*: depthai.VideoEncoder, *bitrateKbps*: int) → None

Set output bitrate in kbps. Final bitrate depends on rate control mode

**setDefaultProfilePreset** (\*args, \*\*kwargs)

Overloaded function.

1. setDefaultProfilePreset(*self*: depthai.VideoEncoder, *width*: int, *height*: int, *fps*: float, *profile*: dai::VideoEncoderProperties::Profile) -> None

Sets a default preset based on specified input size, frame rate and profile

**Parameter width:** Input frame width

**Parameter height:** Input frame height

**Parameter fps:** Frame rate in frames per second

**Parameter profile:** Encoding profile

2. setDefaultProfilePreset(*self*: depthai.VideoEncoder, *size*: Tuple[int, int], *fps*: float, *profile*: dai::VideoEncoderProperties::Profile) -> None

Sets a default preset based on specified input size, frame rate and profile

**Parameter size:** Input frame size

**Parameter fps:** Frame rate in frames per second

**Parameter profile:** Encoding profile

**setFrameRate** (*self*: [depthai.VideoEncoder](#), *frameRate*: *int*) → *None*

Sets expected frame rate

**Parameter frameRate:** Frame rate in frames per second

**setKeyframeFrequency** (*self*: [depthai.VideoEncoder](#), *freq*: *int*) → *None*

Set keyframe frequency. Every Nth frame a keyframe is inserted.

Applicable only to H264 and H265 profiles

Examples:

- 30 FPS video, keyframe frequency: 30. Every 1s a keyframe will be inserted
- 60 FPS video, keyframe frequency: 180. Every 3s a keyframe will be inserted

**setNumBFrames** (*self*: [depthai.VideoEncoder](#), *numBFrames*: *int*) → *None*

Set number of B frames to be inserted

**setNumFramesPool** (*self*: [depthai.VideoEncoder](#), *frames*: *int*) → *None*

Set number of frames in pool

**Parameter frames:** Number of pool frames

**setProfile** (*self*: [depthai.VideoEncoder](#), *width*: *int*, *height*: *int*, *profile*: *dai::VideoEncoderProperties::Profile*) → *None*

Set encoding profile

**setQuality** (*self*: [depthai.VideoEncoder](#), *quality*: *int*) → *None*

Set quality

**Parameter quality:** Value between 0-100%. Approximates quality

**setRateControlMode** (*self*: [depthai.VideoEncoder](#), *mode*: *dai::VideoEncoderProperties::RateControlMode*) → *None*

Set rate control mode

**class** [depthai.VideoEncoderProperties](#)

Bases: [pybind11\\_builtins.pybind11\\_object](#)

Specify VideoEncoder options such as profile, bitrate, ...

**Classes:**

<a href="#">Profile</a>	Encoding profile, H264, H265 or MJPEG
<a href="#">RateControlMode</a>	Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

**Methods:**

<a href="#">__init__</a> (*args, **kwargs)	Initialize self.
--	------------------

**Attributes:**

<a href="#">bitrate</a>
<a href="#">height</a>
<a href="#">keyframeFrequency</a>
<a href="#">maxBitrate</a>
<a href="#">numBFrames</a>
<a href="#">numFramesPool</a>

continues on next page

Table 224 – continued from previous page

*profile**quality**rateCtrlMode**width***class Profile**Bases: `pybind11_builtins.pybind11_object`

Encoding profile, H264, H265 or MJPEG

Members:

H264\_BASELINE

H264\_HIGH

H264\_MAIN

H265\_MAIN

MJPEG

**Attributes:***H264\_BASELINE**H264\_HIGH**H264\_MAIN**H265\_MAIN**MJPEG**name**value***Methods:***\_\_init\_\_*(self, value)

H264\_BASELINE = &lt;Profile.H264\_BASELINE: 0&gt;

H264\_HIGH = &lt;Profile.H264\_HIGH: 1&gt;

H264\_MAIN = &lt;Profile.H264\_MAIN: 2&gt;

H265\_MAIN = &lt;Profile.H265\_MAIN: 3&gt;

MJPEG = &lt;Profile.MJPEG: 4&gt;

*\_\_init\_\_*(self: `depthai.VideoEncoderProperties.Profile`, value: *int*) → None

property name

property value

**class RateControlMode**Bases: `pybind11_builtins.pybind11_object`

Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

Members:

CBR

VBR

**Attributes:**

<i>CBR</i>
<i>VBR</i>
<i>name</i>
<i>value</i>

**Methods:**

<code>__init__(self, value)</code>
------------------------------------

**CBR** = <RateControlMode.CBR: 0>

**VBR** = <RateControlMode.VBR: 1>

`__init__(self: depthai.VideoEncoderProperties.RateControlMode, value: int) → None`

**property name**

**property value**

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**property bitrate**

**property height**

**property keyframeFrequency**

**property maxBitrate**

**property numBFrames**

**property numFramesPool**

**property profile**

**property quality**

**property rateCtrlMode**

**property width**

**class** depthai.XLinkConnection

Bases: pybind11\_builtins.pybind11\_object

**Methods:**

<code>__init__(*args, **kwargs)</code>	Overloaded function.
<code>getAllConnectedDevices(state)</code>	
<code>getDeviceByMxId(mxId, state)</code>	
<code>getFirstDevice(state)</code>	

`__init__(*args, **kwargs)`

Overloaded function.

1. `__init__(self: depthai.XLinkConnection, arg0: depthai.DeviceInfo, arg1: List[int]) -> None`

2. `__init__(self: depthai.XLinkConnection, arg0: depthai.DeviceInfo, arg1: str) -> None`

```
3. __init__(self: depthai.XLinkConnection, arg0: depthai.DeviceInfo) -> None

static getAllConnectedDevices (state: depthai.XLinkDeviceState = <XLinkDeviceState.X_LINK_ANY_STATE: 0>) -> List[depthai.DeviceInfo]

static getDeviceByMxId (mxId: str, state: depthai.XLinkDeviceState = <XLinkDeviceState.X_LINK_ANY_STATE: 0>) -> Tuple[bool, depthai.DeviceInfo]

static getFirstDevice (state: depthai.XLinkDeviceState = <XLinkDeviceState.X_LINK_ANY_STATE: 0>) -> Tuple[bool, depthai.DeviceInfo]
```

```
class depthai.XLinkDeviceState
    Bases: pybind11_builtins.pybind11_object
```

Members:

```
X_LINK_ANY_STATE
X_LINK_BOOTED
X_LINK_UNBOOTED
X_LINK_BOOTLOADER
```

Attributes:

<code>X_LINK_ANY_STATE</code>
<code>X_LINK_BOOTED</code>
<code>X_LINK_BOOTLOADER</code>
<code>X_LINK_UNBOOTED</code>
<code>name</code>
<code>value</code>

Methods:

<code>__init__(self, value)</code>
------------------------------------

```
X_LINK_ANY_STATE = <XLinkDeviceState.X_LINK_ANY_STATE: 0>
X_LINK_BOOTED = <XLinkDeviceState.X_LINK_BOOTED: 1>
X_LINK_BOOTLOADER = <XLinkDeviceState.X_LINK_BOOTLOADER: 3>
X_LINK_UNBOOTED = <XLinkDeviceState.X_LINK_UNBOOTED: 2>

__init__(self: depthai.XLinkDeviceState, value: int) -> None
```

```
property name
property value
```

```
class depthai.XLinkIn
    Bases: depthai.Node
```

XLinkIn node. Receives messages over XLink.

Methods:

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>getMaxDataSize(self)</code>	Get maximum messages size in bytes
<code>getNumFrames(self)</code>	Get number of frames in pool

continues on next page

Table 232 – continued from previous page

<code>getStreamName(self)</code>	Get stream name
<code>setMaxDataSize(self, maxDataSize)</code>	Set maximum message size it can receive
<code>setNumFrames(self, numFrames)</code>	Set number of frames in pool for sending messages forward
<code>setStreamName(self, streamName)</code>	Specifies XLink stream name to use.

**Attributes:**

<code>out</code>	Outputs message of same type as send from host.
------------------	---

`__init__` (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

`getMaxDataSize` (self: `depthai.XLinkIn`) → int

Get maximum messages size in bytes

`getNumFrames` (self: `depthai.XLinkIn`) → int

Get number of frames in pool

`getStreamName` (self: `depthai.XLinkIn`) → str

Get stream name

**property out**

Outputs message of same type as send from host.

`setMaxDataSize` (self: `depthai.XLinkIn`, `maxDataSize`: int) → None

Set maximum message size it can receive

**Parameter maxDataSize:** Maximum size in bytes

`setNumFrames` (self: `depthai.XLinkIn`, `numFrames`: int) → None

Set number of frames in pool for sending messages forward

**Parameter numFrames:** Maximum number of frames in pool

`setStreamName` (self: `depthai.XLinkIn`, `streamName`: str) → None

Specifies XLink stream name to use.

The name should not start with double underscores ‘\_\_’, as those are reserved for internal use.

**Parameter name:** Stream name

**class** `depthai.XLinkOut`

Bases: `depthai.Node`

XLinkOut node. Sends messages over XLink.

**Methods:**

<code>__init__</code> (*args, **kwargs)	Initialize self.
<code>getFpsLimit</code> (self)	Get rate limit in messages per second
<code>getMetadataOnly</code> (self)	Get whether to transfer only messages attributes and not buffer data
<code>getStreamName</code> (self)	Get stream name
<code>setFpsLimit</code> (self, <code>fpsLimit</code> )	Specifies a message sending limit.
<code>setMetadataOnly</code> (self, <code>arg0</code> )	Specify whether to transfer only messages attributes and not buffer data
<code>setStreamName</code> (self, <code>streamName</code> )	Specifies XLink stream name to use.

**Attributes:**

<code>input</code>	Input for any type of messages to be transfered over XLink stream
--------------------	---

`__init__` (\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

`getFpsLimit` (self: `depthai.XLinkOut`) → float

Get rate limit in messages per second

`getMetadataOnly` (self: `depthai.XLinkOut`) → bool

Get whether to transfer only messages attributes and not buffer data

`getStreamName` (self: `depthai.XLinkOut`) → str

Get stream name

**property** `input`

Input for any type of messages to be transfered over XLink stream

Default queue is blocking with size 8

`setFpsLimit` (self: `depthai.XLinkOut`, fpsLimit: float) → None

Specifies a message sending limit. It's approximated from specified rate.

**Parameter fps:** Approximate rate limit in messages per second

`setMetadataOnly` (self: `depthai.XLinkOut`, arg0: bool) → None

Specify whether to transfer only messages attributes and not buffer data

`setStreamName` (self: `depthai.XLinkOut`, streamName: str) → None

Specifies XLink stream name to use.

The name should not start with double underscores '`__`', as those are reserved for internal use.

**Parameter name:** Stream name

**class** `depthai.XLinkPlatform`

Bases: `pybind11_builtins.pybind11_object`

Members:

`X_LINK_ANY_PLATFORM`

`X_LINK_MYRIAD_2`

`X_LINK_MYRIAD_X`

**Attributes:**

`X_LINK_ANY_PLATFORM`

`X_LINK_MYRIAD_2`

`X_LINK_MYRIAD_X`

`name`

`value`

**Methods:**

`__init__` (self, value)

`X_LINK_ANY_PLATFORM = <XLinkPlatform.X_LINK_ANY_PLATFORM: 0>`



```
X_LINK_MYRIAD_2 = <XLinkPlatform.X_LINK_MYRIAD_2: 2450>
```

```
X_LINK_MYRIAD_X = <XLinkPlatform.X_LINK_MYRIAD_X: 2480>
```

```
__init__(self: depthai.XLinkPlatform, value: int) → None
```

property name

property value

```
class depthai.XLinkProtocol
```

Bases: `pybind11_builtins.pybind11_object`

Members:

`X_LINK_USB_VSC`

`X_LINK_USB_CDC`

`X_LINK_PCIE`

`X_LINK_IPC`

`X_LINK_NMB_OF_PROTOCOLS`

`X_LINK_ANY_PROTOCOL`

Attributes:

---

`X_LINK_ANY_PROTOCOL`

---

`X_LINK_IPC`

---

`X_LINK_NMB_OF_PROTOCOLS`

---

`X_LINK_PCIE`

---

`X_LINK_USB_CDC`

---

`X_LINK_USB_VSC`

---

`name`

---

`value`

---

Methods:

---

`__init__(self, value)`

---

```
X_LINK_ANY_PROTOCOL = <XLinkProtocol.X_LINK_ANY_PROTOCOL: 5>
```

```
X_LINK_IPC = <XLinkProtocol.X_LINK_IPC: 3>
```

```
X_LINK_NMB_OF_PROTOCOLS = <XLinkProtocol.X_LINK_NMB_OF_PROTOCOLS: 4>
```

```
X_LINK_PCIE = <XLinkProtocol.X_LINK_PCIE: 2>
```

```
X_LINK_USB_CDC = <XLinkProtocol.X_LINK_USB_CDC: 1>
```

```
X_LINK_USB_VSC = <XLinkProtocol.X_LINK_USB_VSC: 0>
```

```
__init__(self: depthai.XLinkProtocol, value: int) → None
```

property name

property value

```
class depthai.YoloDetectionNetwork
```

Bases: `depthai.DetectionNetwork`

YoloDetectionNetwork node. Parses Yolo results

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setAnchorMasks(self, anchorMasks, List[int])</code>	Set anchor masks
<code>setAnchors(self, anchors)</code>	Set anchors
<code>setCoordinateSize(self, coordinates)</code>	Set coordianate size
<code>setIouThreshold(self, thresh)</code>	Set Iou threshold
<code>setNumClasses(self, numClasses)</code>	Set num classes

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**setAnchorMasks** (*self*: depthai.YoloDetectionNetwork, *anchorMasks*: Dict[str, List[int]]) → None  
Set anchor masks

**setAnchors** (*self*: depthai.YoloDetectionNetwork, *anchors*: List[float]) → None  
Set anchors

**setCoordinateSize** (*self*: depthai.YoloDetectionNetwork, *coordinates*: int) → None  
Set coordianate size

**setIouThreshold** (*self*: depthai.YoloDetectionNetwork, *thresh*: float) → None  
Set Iou threshold

**setNumClasses** (*self*: depthai.YoloDetectionNetwork, *numClasses*: int) → None  
Set num classes

**class** depthai.YoloSpatialDetectionNetwork

Bases: *depthai.SpatialDetectionNetwork*

YoloSpatialDetectionNetwork. (tiny)Yolov3/v4 based network with spatial location data.

**Methods:**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>setAnchorMasks(self, anchorMasks, List[int])</code>	Set anchor masks
<code>setAnchors(self, anchors)</code>	Set anchors
<code>setCoordinateSize(self, coordinates)</code>	Set coordianate size
<code>setIouThreshold(self, thresh)</code>	Set Iou threshold
<code>setNumClasses(self, numClasses)</code>	Set num classes

`__init__(*args, **kwargs)`

Initialize self. See help(type(self)) for accurate signature.

**setAnchorMasks** (*self*: depthai.YoloSpatialDetectionNetwork, *anchorMasks*: Dict[str, List[int]]) → None  
Set anchor masks

**setAnchors** (*self*: depthai.YoloSpatialDetectionNetwork, *anchors*: List[float]) → None  
Set anchors

**setCoordinateSize** (*self*: depthai.YoloSpatialDetectionNetwork, *coordinates*: int) → None  
Set coordianate size

**setIouThreshold** (*self*: depthai.YoloSpatialDetectionNetwork, *thresh*: float) → None  
Set Iou threshold

**setNumClasses** (*self*: depthai.YoloSpatialDetectionNetwork, *numClasses*: int) → None  
Set num classes

## 3.35 C++ API Reference

namespace dai

### Enums

#### enum CameraBoardSocket

Which Camera socket to use.

AUTO denotes that the decision will be made by device

*Values:*

enumerator AUTO

enumerator RGB

enumerator LEFT

enumerator RIGHT

#### enum CameraImageOrientation

Camera sensor image orientation / pixel readout. This exposes direct sensor settings. 90 or 270 degrees rotation is not available.

AUTO denotes that the decision will be made by device (e.g. on OAK-1/megaAI: ROTATE\_180\_DEG).

*Values:*

enumerator AUTO

enumerator NORMAL

enumerator HORIZONTAL\_MIRROR

enumerator VERTICAL\_FLIP

enumerator ROTATE\_180\_DEG

#### enum ProcessorType

On which processor the node will be placed

Enum specifying processor

*Values:*

enumerator LOS

enumerator LRT

#### enum DatatypeEnum

*Values:*

enumerator Buffer

enumerator ImgFrame

enumerator NNData

enumerator ImageManipConfig

enumerator CameraControl

enumerator ImgDetections

enumerator SpatialImgDetections

```
    enumerator SystemInformation
    enumerator SpatialLocationCalculatorConfig
    enumerator SpatialLocationCalculatorData
enum LogLevel
    Values:
    enumerator TRACE
    enumerator DEBUG
    enumerator INFO
    enumerator WARN
    enumerator ERR
    enumerator CRITICAL
    enumerator OFF
```

## Functions

```
bool initialize ()
bool isDatatypeSubclassOf (DatatypeEnum parent, DatatypeEnum children)
```

## Variables

```
constexpr const char *LOG_DEFAULT_PATTERN = "[%E.%e] [%n] [%^%l%$] %v"
constexpr const char *XLINK_CHANNEL_PIPELINE_CONFIG = "__pipeline_config"
constexpr const char *XLINK_CHANNEL_MAIN_RPC = "__rpc_main"
constexpr const char *XLINK_CHANNEL_TIMESYNC = "__timesync"
constexpr const char *XLINK_CHANNEL_LOG = "__log"
constexpr std::uint32_t XLINK_USB_BUFFER_MAX_SIZE = 5 * 1024 * 1024
constexpr const std::chrono::milliseconds XLINK_WATCHDOG_TIMEOUT = {1500}
class ADatatype
    #include <ADatatype.hpp> Abstract message.
    Subclassed by dai::Buffer
struct Asset
    #include <AssetManager.hpp> Asset is identified with string key and can store arbitrary binary data.
class AssetManager
    #include <AssetManager.hpp> AssetManager can store assets and serialize.
```

## Public Functions

void **addExisting** (std::vector<std::shared\_ptr<Asset>> assets)

Adds all assets in an array to the *AssetManager*

### Parameters

- assets: Vector of assets to add

void **add** (Asset asset)

Adds an asset object to *AssetManager*.

### Parameters

- asset: *Asset* to add

void **add** (const std::string &key, Asset asset)

Adds an asset object to *AssetManager* with a specified key. Key value will be assigned to an *Asset* as well

If asset with key already exists, the function throws an error

### Parameters

- key: Key under which the asset should be stored
- asset: *Asset* to store

void **set** (const std::string &key, Asset asset)

Adds or overwrites existing asset with a specified key.

### Parameters

- key: Key under which the asset should be stored
- asset: *Asset* to store

std::shared\_ptr<const Asset> **get** (const std::string &key) const

**Return** *Asset* assigned to the specified key or throws an error otherwise

std::shared\_ptr<Asset> **get** (const std::string &key)

**Return** *Asset* assigned to the specified key or throws an error otherwise

std::vector<std::shared\_ptr<const Asset>> **getAll** () const

**Return** All asset stored in the *AssetManager*

std::vector<std::shared\_ptr<Asset>> **getAll** ()

**Return** All asset stored in the *AssetManager*

std::size\_t **size** () const

**Return** Number of asset stored in the *AssetManager*

void **remove** (const std::string &key)

Removes asset with key

### Parameters

- key: Key of asset to remove

void **serialize** (Assets &serAssets, std::vector<std::uint8\_t> &assetStorage) const

Serializes.

**class Assets**

Subclassed by *dai::AssetsMutable*

**class AssetsMutable** : public *dai::Assets*

**struct AssetView**

**class Buffer** : public *dai::ADatatype*

#include <Buffer.hpp> Base message - buffer of binary data.

Subclassed by *dai::CameraControl*, *dai::ImageManipConfig*, *dai::ImgDetections*, *dai::ImgFrame*, *dai::NNData*, *dai::SpatialImgDetections*, *dai::SpatialLocationCalculatorConfig*, *dai::SpatialLocationCalculatorData*, *dai::SystemInformation*

## Public Functions

**Buffer** ()

Creates *Buffer* message.

std::vector<std::uint8\_t> &**getData** ()

**Return** Reference to internal buffer

void **setData** (std::vector<std::uint8\_t> *data*)

**Parameters**

- *data*: Copies data to internal buffer

**class CallbackHandler**

**class CameraControl** : public *dai::Buffer*

*#include <CameraControl.hpp>* *CameraControl* message Specifies various camera control commands like:

- Still capture
- Auto focus
- Anti banding
- Auto white balance
- Scene
- Effect
- ...

## Public Functions

**CameraControl** ()

Construct *CameraControl* message.

void **setCaptureStill** (bool *capture*)

Set a command to capture a still image

void **setStartStreaming** ()

Set a command to start streaming

void **setStopStreaming** ()

Set a command to stop streaming

void **setAutoFocusMode** (AutoFocusMode *mode*)

Set a command to specify autofocus mode

void **setAutoFocusTrigger** ()

Set a command to trigger autofocus

void **setAutoFocusRegion** (uint16\_t *startX*, uint16\_t *startY*, uint16\_t *width*, uint16\_t *height*)

Set a command to specify focus region in pixels

**Parameters**

- *startX*: X coordinate of top left corner of region

- `startY`: Y coordinate of top left corner of region
- `width`: Region width
- `height`: Region height

void **setManualFocus** (uint8\_t *lensPosition*)

Set a command to specify manual focus position

**Parameters**

- `lensPosition`: specify lens position 0..255

void **setAutoExposureEnable** ()

Set a command to enable auto exposure

void **setAutoExposureLock** (bool *lock*)

Set a command to specify lock auto exposure

**Parameters**

- `lock`: Auto exposure lock mode enabled or disabled

void **setAutoExposureRegion** (uint16\_t *startX*, uint16\_t *startY*, uint16\_t *width*, uint16\_t *height*)

Set a command to specify auto exposure region in pixels

**Parameters**

- `startX`: X coordinate of top left corner of region
- `startY`: Y coordinate of top left corner of region
- `width`: Region width
- `height`: Region height

void **setAutoExposureCompensation** (int8\_t *compensation*)

Set a command to specify auto exposure compensation

**Parameters**

- `compensation`: Compensation value between -128..127

void **setAntiBandingMode** (AntiBandingMode *mode*)

Set a command to specify auto banding mode

**Parameters**

- `mode`: Auto banding mode to use

void **setManualExposure** (uint32\_t *exposureTimeUs*, uint32\_t *sensitivityIso*)

Set a command to manually specify exposure

**Parameters**

- `exposureTimeUs`: Exposure time in microseconds
- `sensitivityIso`: Sensitivity as ISO value

void **setAutoWhiteBalanceMode** (AutoWhiteBalanceMode *mode*)

Set a command to specify auto white balance mode

**Parameters**

- `mode`: Auto white balance mode to use

void **setAutoWhiteBalanceLock** (bool *lock*)

Set a command to specify auto white balance lock

**Parameters**

- `lock`: Auto white balance lock mode enabled or disabled

void **setBrightness** (uint16\_t *value*)

Set a command to specify auto white balance lock

**Parameters**

- `lock`: Auto white balance lock mode enabled or disabled

void **setContrast** (uint16\_t *value*)

Set a command to specify auto white balance lock

**Parameters**

- **lock**: Auto white balance lock mode enabled or disabled

void **setSaturation** (uint16\_t *value*)

Set a command to specify saturation value

**Parameters**

- **value**: Saturation

void **setSharpness** (uint16\_t *value*)

Set a command to specify sharpness value

**Parameters**

- **value**: Sharpness

void **setNoiseReductionStrength** (uint16\_t *value*)

Set a command to specify noise reduction strength

**Parameters**

- **value**: Noise reduction strength

void **setLumaDenoise** (uint16\_t *value*)

Set a command to specify luma denoise value

**Parameters**

- **value**: Luma denoise

void **setChromaDenoise** (uint16\_t *value*)

Set a command to specify chroma denoise value

**Parameters**

- **value**: Chroma denoise

void **setSceneMode** (SceneMode *mode*)

Set a command to specify scene mode

**Parameters**

- **mode**: Scene mode

void **setEffectMode** (EffectMode *mode*)

Set a command to specify effect mode

**Parameters**

- **mode**: Effect mode

bool **getCaptureStill** () **const**

Check whether command to capture a still is set

**Return** True if capture still command is set

**struct** **ChipTemperature**

*#include <ChipTemperature.hpp>* Chip temperature information.

Multiple temperature measurement points and their average

## Public Members

float **css**

CPU Subsystem.

float **mss**

Media Subsystem.

float **upa**

Shave Array.

float **dss**

DRAM Subsystem.



float **average**  
Average of measurements.

**struct ColorCameraProperties**  
*#include <ColorCameraProperties.hpp>* Specify ColorCamera options such as camera ID, ...

## Public Types

**enum SensorResolution**  
Select the camera sensor resolution

*Values:*

**enumerator THE\_1080\_P**

**enumerator THE\_4\_K**

**enumerator THE\_12\_MP**

**enum ColorOrder**  
For 24 bit color these can be either RGB or BGR

*Values:*

**enumerator BGR**

**enumerator RGB**

## Public Members

*CameraBoardSocket* **boardSocket** = *CameraBoardSocket::AUTO*  
Which socket will color camera use

*CameraImageOrientation* **imageOrientation** = *CameraImageOrientation::AUTO*  
Camera sensor image orientation / pixel readout

*ColorOrder* **colorOrder** = *ColorOrder::BGR*  
For 24 bit color these can be either RGB or BGR

bool **interleaved** = true  
Are colors interleaved (R1G1B1, R2G2B2, ...) or planar (R1R2..., G1G2..., B1B2)

bool **fp16** = false  
Are values FP16 type (0.0 - 255.0)

uint32\_t **previewHeight** = 300  
Preview frame output height

uint32\_t **previewWidth** = 300  
Preview frame output width

int32\_t **videoWidth** = AUTO  
Preview frame output width

int32\_t **videoHeight** = AUTO  
Preview frame output height

int32\_t **stillWidth** = AUTO  
Preview frame output width

int32\_t **stillHeight** = AUTO  
Preview frame output height

*SensorResolution* **resolution** = *SensorResolution::THE\_1080\_P*  
Select the camera sensor resolution

float **fps** = 30.0  
Camera sensor FPS

float **sensorCropX** = AUTO  
Initial sensor crop, -1 signifies center crop

bool **inputConfigSync** = false  
Whether to wait for config at 'inputConfig' io

bool **previewKeepAspectRatio** = true  
Whether to keep aspect ratio of input (video size) or not

**struct CpuUsage**  
*#include <CpuUsage.hpp>* *CpuUsage* structure  
Average usage in percent and time span of the average (since last query)

**Public Members**

float **average**  
Average CPU usage, expressed with a normalized value (0-1)

int32\_t **msTime**  
Time span in which the average was calculated in milliseconds.

**class DataInputQueue**  
*#include <DataQueue.hpp>* Access to send messages through XLink stream

### Public Functions

void **setMaxDataSize** (std::size\_t *maxSize*)  
Sets maximum message size. If message is larger than specified, then an exception is issued.

#### Parameters

- *maxSize*: Maximum message size to add to queue

std::size\_t **getMaxDataSize** ()  
Gets maximum queue size.

**Return** Maximum message size

void **setBlocking** (bool *blocking*)  
Sets queue behavior when full (*maxSize*)

#### Parameters

- *blocking*: Specifies if block or overwrite the oldest message in the queue

bool **getBlocking** () **const**  
Gets current queue behavior when full (*maxSize*)

**Return** true if blocking, false otherwise

void **setMaxSize** (unsigned int *maxSize*)  
Sets queue maximum size

#### Parameters

- *maxSize*: Specifies maximum number of messages in the queue

unsigned int **getMaxSize** (unsigned int *maxSize*) **const**

Gets queue maximum size

**Return** Maximum queue size

std::string **getName** () **const**

Gets queues name

**Return** Queue name

void **send** (**const** std::shared\_ptr<*RawBuffer*> &*rawMsg*)

Adds a raw message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

**Parameters**

- *rawMsg*: Message to add to the queue

void **send** (**const** std::shared\_ptr<*ADatatype*> &*msg*)

Adds a message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

**Parameters**

- *msg*: Message to add to the queue

void **send** (**const** *ADatatype* &*msg*)

Adds a message to the queue, which will be picked up and sent to the device. Can either block if 'blocking' behavior is true or overwrite oldest

**Parameters**

- *msg*: Message to add to the queue

bool **send** (**const** std::shared\_ptr<*RawBuffer*> &*rawMsg*, std::chrono::milliseconds *timeout*)

Adds message to the queue, which will be picked up and sent to the device. Can either block until timeout if 'blocking' behavior is true or overwrite oldest

**Parameters**

- *rawMsg*: Message to add to the queue
- *timeout*: Maximum duration to block in milliseconds

bool **send** (**const** std::shared\_ptr<*ADatatype*> &*msg*, std::chrono::milliseconds *timeout*)

Adds message to the queue, which will be picked up and sent to the device. Can either block until timeout if 'blocking' behavior is true or overwrite oldest

**Parameters**

- *msg*: Message to add to the queue
- *timeout*: Maximum duration to block in milliseconds

bool **send** (**const** *ADatatype* &*msg*, std::chrono::milliseconds *timeout*)

Adds message to the queue, which will be picked up and sent to the device. Can either block until timeout if 'blocking' behavior is true or overwrite oldest

**Parameters**

- *msg*: Message to add to the queue
- *timeout*: Maximum duration to block in milliseconds

**class DataOutputQueue**

*#include <DataQueue.hpp>* Access to receive messages coming from XLink stream

## Public Types

**using CallbackId** = int  
Alias for callback id.

## Public Functions

void **setBlocking** (bool *blocking*)  
Sets queue behavior when full (maxSize)

### Parameters

- *blocking*: Specifies if block or overwrite the oldest message in the queue

bool **getBlocking** () **const**  
Gets current queue behavior when full (maxSize)

**Return** true if blocking, false otherwise

void **setMaxSize** (unsigned int *maxSize*)  
Sets queue maximum size

### Parameters

- *maxSize*: Specifies maximum number of messages in the queue

unsigned int **getMaxSize** (unsigned int *maxSize*) **const**  
Gets queue maximum size

**Return** Maximum queue size

std::string **getName** () **const**  
Gets queues name

**Return** Queue name

*CallbackId* **addCallback** (std::function<void> std::string, std::shared\_ptr<*ADatatype*>  
> Adds a callback on message received

**Return** Callback id

### Parameters

- *callback*: Callback function with queue name and message pointer

*CallbackId* **addCallback** (std::function<void> std::shared\_ptr<*ADatatype*>  
> Adds a callback on message received

**Return** Callback id

### Parameters

- *callback*: Callback function with message pointer

*CallbackId* **addCallback** (std::function<void>  
> *callback* Adds a callback on message received

**Return** Callback id

### Parameters

- *callback*: Callback function without any parameters

bool **removeCallback** (*CallbackId* *callbackId*)  
Removes a callback

**Return** true if callback was removed, false otherwise

### Parameters

- *callbackId*: Id of callback to be removed

```

template<class T>
bool has ()
    Check whether front of the queue has message of type T
    Return true if queue isn't empty and the first element is of type T, false otherwise

bool has ()
    Check whether front of the queue has a message (isn't empty)
    Return true if queue isn't empty, false otherwise

template<class T>
std::shared_ptr<T> tryGet ()
    Try to retrieve message T from queue. If message isn't of type T it returns nullptr
    Return Message of type T or nullptr if no message available

std::shared_ptr<ADatatype> tryGet ()
    Try to retrieve message from queue. If no message available, return immediately with nullptr
    Return Message or nullptr if no message available

template<class T>
std::shared_ptr<T> get ()
    Block until a message is available.
    Return Message of type T or nullptr if no message available

std::shared_ptr<ADatatype> get ()
    Block until a message is available.
    Return Message or nullptr if no message available

template<class T>
std::shared_ptr<T> front ()
    Gets first message in the queue.
    Return Message of type T or nullptr if no message available

std::shared_ptr<ADatatype> front ()
    Gets first message in the queue.
    Return Message or nullptr if no message available

template<class T, typename Rep, typename Period>
std::shared_ptr<T> get (std::chrono::duration<Rep, Period> timeout, bool &hasTimedout)
    Block until a message is available with a timeout.
    Return Message of type T otherwise nullptr if message isn't type T or timeout occurred
    Parameters
        • timeout: Duration for which the function should block
        • [out] hasTimedout: Outputs true if timeout occurred, false otherwise

template<typename Rep, typename Period>
std::shared_ptr<ADatatype> get (std::chrono::duration<Rep, Period> timeout, bool &hasTimedout)
    Block until a message is available with a timeout.
    Return Message of type T otherwise nullptr if message isn't type T or timeout occurred
    Parameters
        • timeout: Duration for which the function should block
        • [out] hasTimedout: Outputs true if timeout occurred, false otherwise

template<class T>
std::vector<std::shared_ptr<T>> tryGetAll ()
    Try to retrieve all messages in the queue.

```

**Return** Vector of messages which can either be of type T or nullptr

```
std::vector<std::shared_ptr<ADatatype>> tryGetAll ()
```

Try to retrieve all messages in the queue.

**Return** Vector of messages

```
template<class T>
```

```
std::vector<std::shared_ptr<T>> getAll ()
```

Block until at least one message in the queue. Then return all messages from the queue.

**Return** Vector of messages which can either be of type T or nullptr

```
std::vector<std::shared_ptr<ADatatype>> getAll ()
```

Block until at least one message in the queue. Then return all messages from the queue.

**Return** Vector of messages

```
template<class T, typename Rep, typename Period>
```

```
std::vector<std::shared_ptr<T>> getAll (std::chrono::duration<Rep, Period> timeout, bool &has-  
Timedout)
```

Block for maximum timeout duration. Then return all messages from the queue.

**Return** Vector of messages which can either be of type T or nullptr

**Parameters**

- timeout: Maximum duration to block
- [out] hasTimedout: Outputs true if timeout occurred, false otherwise

```
template<typename Rep, typename Period>
```

```
std::vector<std::shared_ptr<ADatatype>> getAll (std::chrono::duration<Rep, Period> timeout,  
bool &hasTimedout)
```

Block for maximum timeout duration. Then return all messages from the queue.

**Return** Vector of messages

**Parameters**

- timeout: Maximum duration to block
- [out] hasTimedout: Outputs true if timeout occurred, false otherwise

```
struct DetectionNetworkProperties : public dai::NeuralNetworkProperties
```

```
#include <DetectionNetworkProperties.hpp> Properties for DetectionNetwork
```

Subclassed by *dai::SpatialDetectionNetworkProperties*

## Public Members

DetectionNetworkType **nnFamily**

Generic Neural Network properties.

int **classes**

YOLO specific network properties.

```
class Device
```

```
#include <Device.hpp> Represents the DepthAI device with the methods to interact with it.
```

## Public Functions

**Device** (const *Pipeline* &pipeline)

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameters**

- pipeline: - *Pipeline* to be executed on the device

**Device** (const *Pipeline* &pipeline, bool usb2Mode)

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameters**

- pipeline: - *Pipeline* to be executed on the device
- usb2Mode: - Boot device using USB2 mode firmware

**Device** (const *Pipeline* &pipeline, const char \*pathToCmd)

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameters**

- pipeline: - *Pipeline* to be executed on the device
- pathToCmd: - Path to custom device firmware

**Device** (const *Pipeline* &pipeline, const std::string &pathToCmd)

Connects to any available device with a DEFAULT\_SEARCH\_TIME timeout.

**Parameters**

- pipeline: - *Pipeline* to be executed on the device
- pathToCmd: - Path to custom device firmware

**Device** (const *Pipeline* &pipeline, const *DeviceInfo* &devInfo, bool usb2Mode = false)

Connects to device specified by devInfo.

**Parameters**

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- usb2Mode: - Boot device using USB2 mode firmware

**Device** (const *Pipeline* &pipeline, const *DeviceInfo* &devInfo, const char \*pathToCmd)

Connects to device specified by devInfo.

**Parameters**

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- pathToCmd: - Path to custom device firmware

**Device** (const *Pipeline* &pipeline, const *DeviceInfo* &devInfo, const std::string &pathToCmd)

Connects to device specified by devInfo.

**Parameters**

- pipeline: - *Pipeline* to be executed on the device
- devInfo: - *DeviceInfo* which specifies which device to connect to
- usb2Mode: - Path to custom device firmware

**~Device** ()

*Device* destructor. Closes the connection and data queues.

bool **isPipelineRunning** ()

Checks if devices pipeline is already running

**Return** true if running, false otherwise

bool **startPipeline** ()

Starts the execution of the devices pipeline

**Return** true if pipeline started, false otherwise

void **setLogLevel** (*LogLevel level*)

Sets the devices logging severity level. This level affects which logs are transferred from device to host.

**Parameters**

- *level*: Logging severity

*LogLevel* **getLogLevel** ()

Gets current logging severity level of the device.

**Return** Logging severity level

void **setLogOutputLevel** (*LogLevel level*)

Sets logging level which decides printing level to standard output. If lower than setLogLevel, no messages will be printed

**Parameters**

- *level*: - Standard output printing severity

*LogLevel* **getLogOutputLevel** ()

Gets logging level which decides printing level to standard output.

**Return** Standard output printing severity

int **addLogCallback** (std::function<void> *LogMessage*

> *callback*) Add a callback for device logging. The callback will be called from a separate thread with the *LogMessage* being passed.

**Return** Id which can be used to later remove the callback

**Parameters**

- *callback*: - Callback to call whenever a log message arrives

bool **removeLogCallback** (int *callbackId*)

Removes a callback

**Return** true if callback was removed, false otherwise

**Parameters**

- *callbackId*: Id of callback to be removed

void **setSystemInformationLoggingRate** (float *rateHz*)

Sets rate of system information logging (“info” severity). Default 1Hz If parameter is less or equal to zero, then system information logging will be disabled

**Parameters**

- *rateHz*: Logging rate in Hz

float **getSystemInformationLoggingRate** ()

Gets current rate of system information logging (“info” severity) in Hz.

**Return** Logging rate in Hz

std::shared\_ptr<*DataOutputQueue*> **getOutputQueue** (const std::string &*name*)

Gets an output queue corresponding to stream name. If it doesn’t exist it throws

**Return** Smart pointer to *DataOutputQueue*

**Parameters**

- *name*: Queue/stream name, created by XLinkOut node

std::shared\_ptr<*DataOutputQueue*> **getOutputQueue** (const std::string &*name*, unsigned int *maxSize*, bool *blocking* = true)

Gets a queue corresponding to stream name, if it exists, otherwise it throws. Also sets queue options

**Return** Smart pointer to *DataOutputQueue*

**Parameters**



- **name**: Queue/stream name, set in XLinkOut node
- **maxSize**: Maximum number of messages in queue
- **blocking**: Queue behavior once full. True specifies blocking and false overwriting of oldest messages. Default: true

std::vector<std::string> **getOutputQueueNames** () **const**

Get all available output queue names

**Return** Vector of output queue names

std::shared\_ptr<*DataInputQueue*> **getInputQueue** (**const** std::string &name)

Gets an input queue corresponding to stream name. If it doesn't exist it throws

**Return** Smart pointer to *DataInputQueue*

**Parameters**

- **name**: Queue/stream name, set in XLinkIn node

std::shared\_ptr<*DataInputQueue*> **getInputQueue** (**const** std::string &name, unsigned int *maxSize*, bool *blocking* = true)

Gets an input queue corresponding to stream name. If it doesn't exist it throws. Also sets queue options

**Return** Smart pointer to *DataInputQueue*

**Parameters**

- **name**: Queue/stream name, set in XLinkOut node
- **maxSize**: Maximum number of messages in queue
- **blocking**: Queue behavior once full. True: blocking, false: overwriting of oldest messages. Default: true

std::vector<std::string> **getInputQueueNames** () **const**

Get all available input queue names

**Return** Vector of input queue names

std::vector<std::string> **getQueueEvents** (**const** std::vector<std::string> &queueNames, std::size\_t *maxNumEvents* = std::numeric\_limits<std::size\_t>::max(), std::chrono::microseconds *timeout* = std::chrono::microseconds(-1))

Gets or waits until any of specified queues has received a message

**Return** Names of queues which received messages first

**Parameters**

- **queueNames**: Names of queues for which to block
- **maxNumEvents**: Maximum number of events to remove from queue - Default is unlimited
- **timeout**: Timeout after which return regardless. If negative then wait is indefinite - Default is -1

std::vector<std::string> **getQueueEvents** (std::string *queueName*, std::size\_t *maxNumEvents* = std::numeric\_limits<std::size\_t>::max(), std::chrono::microseconds *timeout* = std::chrono::microseconds(-1))

Gets or waits until specified queue has received a message

**Return** Names of queues which received messages first

**Parameters**

- **queueName**: Name of queues for which to wait for
- **maxNumEvents**: Maximum number of events to remove from queue. Default is unlimited
- **timeout**: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::vector<std::string> getQueueEvents (std::size_t          maxNumEvents          =  
                                         std::numeric_limits<std::size_t>::max(),  
                                         std::chrono::microseconds timeout          =  
                                         std::chrono::microseconds(-1))
```

Gets or waits until any any queue has received a message

**Return** Names of queues which received messages first

**Parameters**

- **maxNumEvents**: Maximum number of events to remove from queue. Default is unlimited
- **timeout**: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (const          std::vector<std::string>          &queueNames,  
                           std::chrono::microseconds timeout = std::chrono::microseconds(-1))
```

Gets or waits until any of specified queues has received a message

**Return** Queue name which received a message first

**Parameters**

- **queueNames**: Names of queues for which to wait for
- **timeout**: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (std::string queueName, std::chrono::microseconds timeout =  
                           std::chrono::microseconds(-1))
```

Gets or waits until specified queue has received a message

**Return** Queue name which received a message

**Parameters**

- **queueNames**: Name of queues for which to wait for
- **timeout**: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

```
std::string getQueueEvent (std::chrono::microseconds timeout = std::chrono::microseconds(-1))
```

Gets or waits until any queue has received a message

**Return** Queue name which received a message

**Parameters**

- **timeout**: Timeout after which return regardless. If negative then wait is indefinite. Default is -1

*MemoryInfo* **getDdrMemoryUsage** ()

Retrieves current DDR memory information from device

**Return** Used, remaining and total ddr memory

*MemoryInfo* **getCmxMemoryUsage** ()

Retrieves current CMX memory information from device

**Return** Used, remaining and total cmx memory

*MemoryInfo* **getLeonCssHeapUsage** ()

Retrieves current CSS Leon CPU heap information from device

**Return** Used, remaining and total heap memory

*MemoryInfo* **getLeonMssHeapUsage** ()

Retrieves current MSS Leon CPU heap information from device

**Return** Used, remaining and total heap memory

*ChipTemperature* **getChipTemperature** ()

Retrieves current chip temperature as measured by device

**Return** Temperature of various onboard sensors

*CpuUsage* **getLeonCssCpuUsage** ()

Retrieves average CSS Leon CPU usage

**Return** Average CPU usage and sampling duration

*CpuUsage* **getLeonMssCpuUsage** ()

Retrieves average MSS Leon CPU usage

**Return** Average CPU usage and sampling duration

void **close** ()

Explicitly closes connection to device.

**Note** This function does not need to be explicitly called as destructor closes the device automatically

bool **isClosed** () **const**

Is the device already closed (or disconnected)

## Public Static Functions

template<typename **Rep**, typename **Period**>

std::tuple<bool, *DeviceInfo*> **getAnyAvailableDevice** (std::chrono::duration<*Rep*, *Period*>  
timeout)

Waits for any available device with a timeout

**Return** a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

**Parameters**

- timeout: - duration of time to wait for the any device

std::tuple<bool, *DeviceInfo*> **getAnyAvailableDevice** ()

Gets any available device

**Return** a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

std::tuple<bool, *DeviceInfo*> **getFirstAvailableDevice** ()

Gets first available device. *Device* can be either in XLINK\_UNBOOTED or XLINK\_BOOTLOADER state

**Return** a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

std::tuple<bool, *DeviceInfo*> **getDeviceByMxId** (std::string *mxId*)

Finds a device by MX ID. Example: 14442C10D13EABCE00

**Return** a tuple of bool and *DeviceInfo*. Bool specifies if device was found. *DeviceInfo* specifies the found device

**Parameters**

- *mxId*: - MyraidX ID which uniquely specifies a device

std::vector<*DeviceInfo*> **getAllAvailableDevices** ()

Returns all connected devices

**Return** vector of connected devices

std::vector<std::uint8\_t> **getEmbeddedDeviceBinary** (bool *usb2Mode*, *Open-*  
*VINO*::Version *version* =

*Pipeline*::DEFAULT\_OPENVINO\_VERSION)

Gets device firmware binary for a specific *OpenVINO* version

**Return** firmware binary

**Parameters**

- `usb2Mode`: - USB2 mode firmware
- `version`: - Version of *OpenVINO* which firmware will support

### Public Static Attributes

**constexpr** std::chrono::seconds **DEFAULT\_SEARCH\_TIME** = {3}

Default search time for constructors which discover devices.

**constexpr** std::size\_t **EVENT\_QUEUE\_MAXIMUM\_SIZE** = {2048}

Maximum number of elements in event queue.

**constexpr** float **DEFAULT\_SYSTEM\_INFORMATION\_LOGGING\_RATE\_HZ** = {1.0f}

Default rate at which system information is logged.

**class DeviceBootloader**

*#include <DeviceBootloader.hpp>* Represents the DepthAI bootloader with the methods to interact with it.

### Public Functions

**DeviceBootloader** (const *DeviceInfo* &devInfo)

Connects to or boots device in bootloader mode depending on devInfo state.

#### Parameters

- devInfo: *DeviceInfo* of which to boot or connect to

**DeviceBootloader** (const *DeviceInfo* &devInfo, const std::string &pathToBootloader)

Connects to or boots device in bootloader mode depending on devInfo state with a custom bootloader firmware.

#### Parameters

- devInfo: *DeviceInfo* of which to boot or connect to
- pathToBootloader: Custom bootloader firmware to boot

**DeviceBootloader** (const *DeviceInfo* &devInfo, const char \*pathToBootloader)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

std::tuple<bool, std::string> **flash** (std::function<void> float  
> progressCallback, *Pipeline* &pipelineFlashes a give pipeline to the board.

#### Parameters

- progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress
- pipeline: *Pipeline* to flash to the board

std::tuple<bool, std::string> **flashDepthaiApplicationPackage** (std::function<void> float  
> progressCallback, std::vector<uint8\_t> packageFlashes a specific depthai application package that was generated using createDepthaiApplicationPackage or saveDepthaiApplicationPackage

#### Parameters

- progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress
- package: Depthai application package to flash to the board

std::tuple<bool, std::string> **flashBootloader** (std::function<void> float  
> progressCallback, std::string path = ""Flashes bootloader to the current board

#### Parameters

- progressCallback: Callback that sends back a value between 0..1 which signifies current flashing progress
- path: Optional parameter to custom bootloader to flash

*Version* **getVersion()**

**Return** *Version* of current running bootloader

bool **isEmbeddedVersion()**

**Return** True whether the bootloader running is flashed or booted by library

void **close()**

Explicitly closes connection to device.

**Note** This function does not need to be explicitly called as destructor closes the device automatically

bool **isClosed() const**

Is the device already closed (or disconnected)

## Public Static Functions

std::tuple<bool, *DeviceInfo*> **getFirstAvailableDevice()**

Searches for connected devices in either UNBOOTED or BOOTLOADER states and returns first available.

**Return** Tuple of boolean and *DeviceInfo*. If found boolean is true and *DeviceInfo* describes the device. Otherwise false

std::vector<*DeviceInfo*> **getAllAvailableDevices()**

Searches for connected devices in either UNBOOTED or BOOTLOADER states.

**Return** Vector of all found devices

std::vector<uint8\_t> **createDepthaiApplicationPackage** (*Pipeline* &pipeline, std::string pathToCmd = "")

Creates application package which can be flashed to depthai device.

**Return** Depthai application package

**Parameters**

- pipeline: *Pipeline* from which to create the application package
- pathToCmd: Optional path to custom device firmware

void **saveDepthaiApplicationPackage** (std::string path, *Pipeline* &pipeline, std::string pathToCmd = "")

Saves application package to a file which can be flashed to depthai device.

**Parameters**

- path: Path where to save the application package
- pipeline: *Pipeline* from which to create the application package
- pathToCmd: Optional path to custom device firmware

*Version* **getEmbeddedBootloaderVersion()**

**Return** Embedded bootloader version

std::vector<std::uint8\_t> **getEmbeddedBootloaderBinary()**

**Return** Embedded bootloader binary

**struct Version**

#include <DeviceBootloader.hpp> Bootloader version structure.

## Public Functions

**Version** (const std::string &*v*)  
Construct *Version* from string.

**Version** (unsigned *major*, unsigned *minor*, unsigned *patch*)  
Construct *Version* major, minor and patch numbers.

std::string **toString** () const  
Convert *Version* to string.

**struct DeviceInfo**

*#include <XLinkConnection.hpp>* Describes a connected device

**struct GlobalProperties**

*#include <GlobalProperties.hpp>* Specify properties which apply for whole pipeline

## Public Members

double **leonCssFrequencyHz** = 700 \* 1000 \* 1000  
Set frequency of Leon OS - Increasing can improve performance, at the cost of higher power draw

double **leonMssFrequencyHz** = 700 \* 1000 \* 1000  
Set frequency of Leon RT - Increasing can improve performance, at the cost of higher power draw

**class ImageManipConfig** : public dai::Buffer

*#include <ImageManipConfig.hpp>* *ImageManipConfig* message. Specifies image manipulation options like:

- Crop
- Resize
- Warp
- ...

## Public Functions

**ImageManipConfig** ()  
Construct *ImageManipConfig* message.

void **setCropRect** (float *xmin*, float *ymin*, float *xmax*, float *ymax*)  
Specifies crop with rectangle with normalized values (0..1)

### Parameters

- *xmin*: Top left X coordinate of rectangle
- *ymin*: Top left Y coordinate of rectangle
- *xmax*: Bottom right X coordinate of rectangle
- *ymax*: Bottom right Y coordinate of rectangle

void **setCropRotatedRect** (*RotatedRect* *rr*, bool *normalizedCoords* = true)  
Specifies crop with rotated rectangle. Optionally as non normalized coordinates

### Parameters

- *rr*: Rotated rectangle which specifies crop
- *normalizedCoords*: If true coordinates are in normalized range (0..1) otherwise absolute

void **setCenterCrop** (float *ratio*, float *whRatio* = 1.0f)  
Specifies a centered crop.

**Parameters**

- `ratio`: Ratio between input image and crop region (0..1)
- `whRatio`: Crop region aspect ratio - 1 equals to square, 1.7 equals to 16:9, ...

void **setWarpTransformFourPoints** (std::vector<[Point2f](#)> *pt*, bool *normalizedCoords*)

Specifies warp by suppling 4 points in either absolute or normalized coordinates

**Parameters**

- `pt`: 4 points specifying warp
- `normalizedCoords`: If true `pt` is interpreted as normalized, absolute otherwise

void **setWarpTransformMatrix3x3** (std::vector<float> *mat*)

Specifies warp with a 3x3 matrix

**Parameters**

- `mat`: 3x3 matrix

void **setWarpBorderReplicatePixels** ()

Specifies that warp replicates border pixels

void **setWarpBorderFillColor** (int *red*, int *green*, int *blue*)

Specifies fill color for border pixels. Example:

- `setWarpBorderFillColor(255,255,255)` -> white
- `setWarpBorderFillColor(0,0,255)` -> blue

**Parameters**

- `red`: Red component
- `green`: Green component
- `blue`: Blue component

void **setRotationDegrees** (float *deg*)

Specifies clockwise rotation in degrees

**Parameters**

- `deg`: Rotation in degrees

void **setRotationRadians** (float *rad*)

Specifies clockwise rotation in radians

**Parameters**

- `rad`: Rotation in radians

void **setResize** (int *w*, int *h*)

Specifies output image size. After crop stage the image will be streched to fit.

**Parameters**

- `w`: Width in pixels
- `h`: Height in pixels

void **setResizeThumbnail** (int *w*, int *h*, int *bgRed* = 0, int *bgGreen* = 0, int *bgBlue* = 0)

Specifies output image size. After crop stage the image will be resized by preserving aspect ration. Optionally background can be specified.

**Parameters**

- `w`: Width in pixels
- `h`: Height in pixels
- `bgRed`: Red component
- `bgGreen`: Green component
- `bgBlue`: Blue component

void **setFrameType** ([ImgFrame::Type](#) *name*)

Specify output frame type.

**Parameters**

- `name`: Frame type

```
void setHorizontalFlip (bool flip)  
    Specify horizontal flip  
    Parameters  
        • flip: True to enable flip, false otherwise  
  
void setReusePreviousImage (bool reuse)  
    Instruct ImageManip to not remove current image from its queue and use the same for next message.  
    Parameters  
        • reuse: True to enable reuse, false otherwise  
  
void setSkipCurrentImage (bool skip)  
    Instructs ImageManip to skip current image and wait for next in queue.  
    Parameters  
        • skip: True to skip current image, false otherwise  
  
void setKeepAspectRatio (bool keep)  
    Specifies to whether to keep aspect ratio or not  
  
float getCropXMin () const  
    Return Top left X coordinate of crop region  
  
float getCropYMin () const  
    Return Top left Y coordinate of crop region  
  
float getCropXMax () const  
    Return Bottom right X coordinate of crop region  
  
float getCropYMax () const  
    Return Bottom right Y coordinate of crop region  
  
int getResizeWidth () const  
    Return Output image width  
  
int getResizeHeight () const  
    Return Output image height  
  
bool isResizeThumbnail () const  
    Return True if resize thumbnail mode is set, false otherwise  
  
struct ImageManipProperties  
    #include <ImageManipProperties.hpp> Specify ImageManip options
```

## Public Members

```
RawImageManipConfig initialConfig  
    Initial configuration for ImageManip node.  
  
bool inputConfigSync = false  
    Whether to wait for config at 'inputConfig' IO.  
  
int outputFrameSize = 1 * 1024 * 1024  
    Maximum output frame size in bytes (eg: 300x300 BGR image -> 300*300*3 bytes)  
  
int numFramesPool = 4  
    Num frames in output pool.  
  
struct ImgDetection  
    Subclassed by dai::SpatialImgDetection  
  
class ImgDetections : public dai::Buffer  
    #include <ImgDetections.hpp> ImgDetections message. Carries normalized detection results
```



## Public Functions

### **ImgDetections ()**

Construct *ImgDetections* message.

## Public Members

`std::vector<ImgDetection> &detections`  
Detections.

**class** **ImgFrame** : **public** *dai::Buffer*

*#include* <ImgFrame.hpp> *ImgFrame* message. Carries image data and metadata.

## Public Functions

### **ImgFrame ()**

Construct *ImgFrame* message. *Timestamp* is set to now

`std::chrono::time_point<std::chrono::steady_clock, std::chrono::steady_clock::duration> getTimestamp ()` **const**

Retrieves image timestamp related to *steady\_clock* / *time.monotonic*

`unsigned int getInstanceNum ()` **const**  
Retrieves instance number

`unsigned int getCategory ()` **const**  
Retrieves image category

`unsigned int getSequenceNum ()` **const**  
Retrieves image sequence number

`unsigned int getWidth ()` **const**  
Retrieves image width in pixels

`unsigned int getHeight ()` **const**  
Retrieves image height in pixels

Type `getType ()` **const**  
Retrieves image type

`void setTimestamp (std::chrono::time_point<std::chrono::steady_clock, std::chrono::steady_clock::duration> timestamp)`  
Specifies current timestamp, related to *steady\_clock* / *time.monotonic*

`void setInstanceNum (unsigned int instance)`  
Instance number relates to the origin of the frame (which camera)

#### **Parameters**

- *instance*: Instance number

`void setCategory (unsigned int category)`

#### **Parameters**

- *category*: Image category

`void setSequenceNum (unsigned int seq)`  
Specifies sequence number

#### **Parameters**

- *seq*: Sequence number

void **setWidth** (unsigned int *width*)  
Specifies frame width

**Parameters**

- *width*: frame width

void **setHeight** (unsigned int)  
Specifies frame height

**Parameters**

- *width*: frame height

void **setType** (Type *type*)  
Specifies frame type, RGB, BGR, ...

**Parameters**

- *type*: Type of image

void **setFrame** (cv::Mat *frame*)

Copies cv::Mat data to *ImgFrame* buffer

**Note** This API only available if OpenCV support enabled

**Parameters**

- *frame*: Input cv::Mat frame from which to copy the data

cv::Mat **getFrame** (bool *copy* = false)

Retrieves data as cv::Mat with specified width, height and type

**Note** This API only available if OpenCV support enabled

**Return** cv::Mat with corresponding to *ImgFrame* parameters

**Parameters**

- *copy*: If false only a reference to data is made, otherwise a copy

cv::Mat **getCvFrame** ()

Retrieves cv::Mat suitable for use in common opencv functions. *ImgFrame* is converted to color BGR interleaved or grayscale depending on type.

**Note** This API only available if OpenCV support enabled

A copy is always made

**Return** cv::Mat for use in opencv functions

```
template<typename T>
```

```
class LockingQueue
```

```
struct LogMessage
```

```
struct MemoryInfo
```

```
    #include <MemoryInfo.hpp> MemoryInfo structure
```

```
    Free, remaining and total memory stats
```

```
struct MonoCameraProperties
```

```
    #include <MonoCameraProperties.hpp> Specify MonoCamera options such as camera ID, ...
```

## Public Types

### **enum SensorResolution**

Select the camera sensor resolution: 1280×720, 1280×800, 640×400

*Values:*

**enumerator** *THE\_720\_P*

**enumerator** *THE\_800\_P*

**enumerator** *THE\_400\_P*

## Public Members

*CameraBoardSocket* **boardSocket** = *CameraBoardSocket::AUTO*

Which socket will mono camera use

*CameraImageOrientation* **imageOrientation** = *CameraImageOrientation::AUTO*

Camera sensor image orientation / pixel readout

*SensorResolution* **resolution** = *SensorResolution::THE\_720\_P*

Select the camera sensor resolution

float **fps** = 30.0

Camera sensor FPS

### **struct MyConsumerProperties**

*#include <MyConsumerProperties.hpp>* Specify message and processor placement of MyConsumer node

## Public Members

*ProcessorType* **processorPlacement**

On which processor the node will be placed

### **struct MyProducerProperties**

*#include <MyProducerProperties.hpp>* Specify message and processor placement of MyProducer node

## Public Members

tl::optional<std::string> **message**

Message to be sent forward

*ProcessorType* **processorPlacement** = *ProcessorType::LOS*

On which processor the node will be placed

### **struct NeuralNetworkProperties**

*#include <NeuralNetworkProperties.hpp>* Specify NeuralNetwork options such as blob path, ...

Subclassed by *dai::DetectionNetworkProperties*

## Public Members

tl::optional<std::uint32\_t> **blobSize**

Blob binary size in bytes

std::string **blobUri**

Uri which points to blob

std::uint32\_t **numFrames** = 8

Number of available output tensors in pool

std::uint32\_t **numThreads** = 0

Number of threads to create for running inference. 0 = auto

std::uint32\_t **numNCEPerThread** = 0

Number of NCE (Neural Compute Engine) per inference thread. 0 = auto

**class NNData : public dai::Buffer**

*#include <NNData.hpp>* *NNData* message. Carries tensors and their metadata

## Public Functions

**NNData ()**

Construct *NNData* message.

void **setLayer** (const std::string &name, std::vector<std::uint8\_t> data)

Set a layer with datatype U8.

### Parameters

- name: Name of the layer
- data: Data to store

void **setLayer** (const std::string &name, const std::vector<int> &data)

Set a layer with datatype U8. Integers are casted to bytes.

### Parameters

- name: Name of the layer
- data: Data to store

void **setLayer** (const std::string &name, std::vector<float> data)

Set a layer with datatype FP16. Float values are converted to FP16.

### Parameters

- name: Name of the layer
- data: Data to store

void **setLayer** (const std::string &name, std::vector<double> data)

Set a layer with datatype FP16. Double values are converted to FP16.

### Parameters

- name: Name of the layer
- data: Data to store

std::vector<std::string> **getAllLayerNames () const**

**Return** Names of all layers added

std::vector<*TensorInfo*> **getAllLayers () const**

**Return** All layers and their information

bool **getLayer** (const std::string &name, *TensorInfo* &tensor) const

Retrieve layers tensor information

**Return** True if layer exists, false otherwise

### Parameters

- name: Name of the layer
- [out] tensor: Outputs tensor information of that layer

bool **hasLayer** (const std::string &name) const

Checks if given layer exists

**Return** True if layer exists, false otherwise

**Parameters**

- name: Name of the layer

bool **getLayerDatatype** (const std::string &name, *TensorInfo::DataType* &datatype) const

Retrieve datatype of a layers tensor

**Return** True if layer exists, false otherwise

**Parameters**

- name: Name of the layer
- [out] datatype: Datatype of layers tensor

std::vector<std::uint8\_t> **getLayerUInt8** (const std::string &name) const

Convenience function to retrieve U8 data from layer

**Return** U8 binary data

**Parameters**

- name: Name of the layer

std::vector<float> **getLayerFp16** (const std::string &name) const

Convenience function to retrieve float values from layers FP16 tensor

**Return** Float data

**Parameters**

- name: Name of the layer

std::vector<std::int32\_t> **getLayerInt32** (const std::string &name) const

Convenience function to retrieve INT32 values from layers tensor

**Return** INT32 data

**Parameters**

- name: Name of the layer

std::vector<std::uint8\_t> **getFirstLayerUInt8** () const

Convenience function to retrieve U8 data from first layer

**Return** U8 binary data

std::vector<float> **getFirstLayerFp16** () const

Convenience function to retrieve float values from first layers FP16 tensor

**Return** Float data

std::vector<std::int32\_t> **getFirstLayerInt32** () const

Convenience function to retrieve INT32 values from first layers tensor

**Return** INT32 data

**class Node**

*#include <Node.hpp>* Abstract *Node*.

Subclassed by *dai::node::ColorCamera*, *dai::node::ImageManip*, *dai::node::MonoCamera*, *dai::node::MyProducer*, *dai::node::NeuralNetwork*, *dai::node::SpatialLocationCalculator*, *dai::node::SPIOut*, *dai::node::StereoDepth*, *dai::node::SystemLogger*, *dai::node::VideoEncoder*, *dai::node::XLinkIn*, *dai::node::XLinkOut*

## Public Types

**using Id** = std::int64\_t  
*Node* identifier. Unique for every node on a single *Pipeline*.

## Public Functions

std::string **getName** () **const** = 0  
Retrieves nodes name.

std::vector<Output> **getOutputs** () = 0  
Retrieves all nodes outputs.

std::vector<Input> **getInputs** () = 0  
Retrieves all nodes inputs.

std::vector<std::shared\_ptr<Asset>> **getAssets** ()  
Retrieves all nodes assets.

## Public Members

**const Id id**  
Id of node.

**struct Connection**  
*#include <Node.hpp>* *Connection* between an Input and Output.

**struct NodeConnectionSchema**  
*#include <NodeConnectionSchema.hpp>* Specifies a connection between nodes IOs

**struct NodeIoInfo**

**struct NodeObjInfo**

**class OpenVINO**  
*#include <OpenVINO.hpp>* Support for basic *OpenVINO* related actions like version identification of neural network blobs,...

## Public Types

**enum Version**  
*OpenVINO* Version supported version information.

*Values:*

**enumerator VERSION\_2020\_1**

**enumerator VERSION\_2020\_2**

**enumerator VERSION\_2020\_3**

**enumerator VERSION\_2020\_4**

**enumerator VERSION\_2021\_1**

**enumerator VERSION\_2021\_2**

## Public Static Functions

`std::vector<Version> getVersions ()`

**Return** Supported versions

`std::string getVersionName (Version version)`

Returns string representation of a given version

**Return** Name of a given version

**Parameters**

- version: *OpenVINO* version

*Version* `parseVersionName (const std::string &versionString)`

Creates Version from string representation. Throws if not possible.

**Return** Version object if successful

**Parameters**

- versionString: Version as string

`std::vector<Version> getBlobSupportedVersions (std::uint32_t majorVersion, std::uint32_t minorVersion)`

Returns a list of potentially supported versions for a specified blob major and minor versions.

**Return** Vector of potentially supported versions

**Parameters**

- majorVersion: Major version from *OpenVINO* blob
- minorVersion: Minor version from *OpenVINO* blob

*Version* `getBlobLatestSupportedVersion (std::uint32_t majorVersion, std::uint32_t minorVersion)`

Returns latest potentially supported version by a given blob version.

**Return** Latest potentially supported version

**Parameters**

- majorVersion: Major version from *OpenVINO* blob
- minorVersion: Minor version from *OpenVINO* blob

`bool areVersionsBlobCompatible (Version v1, Version v2)`

Checks whether two blob versions are compatible

`template<typename T>`

`class Pimpl`

`class Pipeline`

*#include <Pipeline.hpp>* Represents the pipeline, set of nodes and connections between them.

## Public Functions

`Pipeline ()`

Constructs a new pipeline

*GlobalProperties* `getGlobalProperties () const`

**Return** Global properties of current pipeline

*PipelineSchema* `getPipelineSchema ()`

**Return** *Pipeline* schema

`template<class N>`

`std::shared_ptr<N> create ()`

Adds a node to pipeline.

*Node* is specified by template argument N

`void remove (std::shared_ptr<Node> node)`  
Removes a node from pipeline.

`std::vector<std::shared_ptr<const Node>> getAllNodes () const`  
Get a vector of all nodes.

`std::vector<std::shared_ptr<Node>> getAllNodes ()`  
Get a vector of all nodes.

`std::shared_ptr<const Node> getNode (Node::Id id) const`  
Get node with id if it exists, nullptr otherwise.

`std::shared_ptr<Node> getNode (Node::Id id)`  
Get node with id if it exists, nullptr otherwise.

`std::vector<Node::Connection> getConnections () const`  
Get all connections.

`const NodeConnectionMap &getConnectionMap () const`  
Get a reference to internal connection representation.

`const NodeMap &getNodeMap () const`  
Get a reference to internal node map.

`void link (const Node::Output &out, const Node::Input &in)`  
Link output to an input. Both nodes must be on the same pipeline  
Throws an error if they aren't or cannot be connected

**Parameters**

- `out`: Nodes output to connect from
- `in`: Nodes input to connect to

`void unlink (const Node::Output &out, const Node::Input &in)`  
Unlink output from an input.  
Throws an error if link doesn't exists

**Parameters**

- `out`: Nodes output to unlink from
- `in`: Nodes input to unlink to

`AssetManager getAllAssets () const`  
Get assets on the pipeline includes nodes assets.

`const AssetManager &getAssetManager () const`  
Get pipelines *AssetManager* as reference.

`AssetManager &getAssetManager ()`  
Get pipelines *AssetManager* as reference.

`void setOpenVINOVersion (OpenVINO::Version version)`  
Set a specific *OpenVINO* version to use with this pipeline.



## Public Static Attributes

```
constexpr auto DEFAULT_OPENVINO_VERSION = PipelineImpl::DEFAULT_OPENVINO_VERSION
    Default Pipeline openvino version.

class PipelineImpl

struct PipelineSchema
    #include <PipelineSchema.hpp> Specifies whole pipeline, nodes, properties and connections between
    nodes IOs

struct Point2f
    #include <Point2f.hpp> Point2f structure
    x and y coordinates that define a 2D point.

struct Point3f
    #include <Point3f.hpp> Point3f structure
    x,y,z coordinates that define a 3D point.

struct RawBuffer
    Subclassed by dai::RawCameraControl, dai::RawImageManipConfig,
    dai::RawImgDetections, dai::RawImgFrame, dai::RawNNData, dai::RawSpatialImgDetections,
    dai::RawSpatialLocationCalculatorConfig, dai::RawSpatialLocations, dai::RawSystemInformation

struct RawCameraControl : public dai::RawBuffer
```

## Public Members

```
uint8_t lensPosition = 0
    Lens/VCM position, range: 0..255. Used with autoFocusMode = OFF. With current IMX378
    modules:
    • max 255: macro focus, at 8cm distance
    • infinite focus at about 120..130 (may vary from module to module)
    • lower values lead to out-of-focus (lens too close to the sensor array)

struct ManualExposureParams

struct RegionParams

struct RawImageManipConfig : public dai::RawBuffer

    struct CropConfig
    struct CropRect
    struct FormatConfig
    struct ResizeConfig
```

## Public Members

```
bool keepAspectRatio = true
    Whether to keep aspect ratio of input or not

struct RawImgDetections : public dai::RawBuffer

struct RawImgFrame : public dai::RawBuffer

    struct Specs

struct RawNNData : public dai::RawBuffer

struct RawSpatialImgDetections : public dai::RawBuffer

struct RawSpatialLocationCalculatorConfig : public dai::RawBuffer

struct RawSpatialLocations : public dai::RawBuffer

struct RawSystemInformation : public dai::RawBuffer
    #include <RawSystemInformation.hpp> System information of device
    Memory usage, cpu usage and chip temperature
```

## Public Members

*MemoryInfo* **ddrMemoryUsage**  
DDR memory usage.

*MemoryInfo* **cmxMemoryUsage**  
CMX memory usage.

*MemoryInfo* **leonCssMemoryUsage**  
LeonCss heap usage.

*MemoryInfo* **leonMssMemoryUsage**  
LeonMss heap usage.

*CpuUsage* **leonCssCpuUsage**  
LeonCss cpu usage.

*CpuUsage* **leonMssCpuUsage**  
LeonMss cpu usage.

*ChipTemperature* **chipTemperature**  
Chip temperatures.

```
struct Rect
    #include <Rect.hpp> Rect structure

    x,y coordinates together with width and height that define a rectangle. Can be either normalized [0,1] or
    absolute representation.
```

## Public Functions

*Point2f* **topLeft** () **const**

The top-left corner.

*Point2f* **bottomRight** () **const**

The bottom-right corner

*Size2f* **size** () **const**

Size (width, height) of the rectangle

float **area** () **const**

Area (width\*height) of the rectangle

bool **empty** () **const**

True if rectangle is empty.

bool **contains** (const *Point2f* &pt) **const**

Checks whether the rectangle contains the point.

bool **isNormalized** () **const**

Whether rectangle is normalized (coordinates in [0,1] range) or not.

*Rect* **denormalize** (int width, int height)

Denormalize rectangle.

**Parameters**

- width: Destination frame width.
- height: Destination frame height.

*Rect* **normalize** (int width, int height)

Normalize rectangle.

**Parameters**

- width: Source frame width.
- height: Source frame height.

**struct RotatedRect**

## Public Members

float **angle**

degrees, increasing clockwise

**struct Size2f**

**struct SpatialDetectionNetworkProperties** : public *dai::DetectionNetworkProperties*

*#include <SpatialDetectionNetworkProperties.hpp>* Properties for SpatialDetectionNetwork

**struct SpatialImgDetection** : public *dai::ImgDetection*

*#include <RawSpatialImgDetections.hpp>* Spatial image detection structure

Contains image detection results together with spatial location data.

**class SpatialImgDetections** : public *dai::Buffer*

*#include <SpatialImgDetections.hpp>* *SpatialImgDetections* message. Carries detection results together with spatial location data

## Public Functions

**SpatialImgDetections ()**

Construct *SpatialImgDetections* message.

## Public Members

std::vector<*SpatialImgDetection*> &**detections**

Detection results.

**class SpatialLocationCalculatorConfig : public dai::Buffer**

*#include <SpatialLocationCalculatorConfig.hpp>* *SpatialLocationCalculatorConfig* message. Carries ROI (region of interest) and threshold for depth calculation

## Public Functions

**SpatialLocationCalculatorConfig ()**

Construct *SpatialLocationCalculatorConfig* message.

void **setROIs** (std::vector<*SpatialLocationCalculatorConfigData*> *ROIs*)

Set a vector of ROIs as configuration data.

**Parameters**

- *ROIs*: Vector of configuration parameters for ROIs (region of interests)

void **addROI** (*SpatialLocationCalculatorConfigData* &*ROI*)

Add a new ROI to configuration data.

**Parameters**

- *roi*: Configuration parameters for ROI (region of interest)

std::vector<*SpatialLocationCalculatorConfigData*> **getConfigData () const**

Retrieve configuration data for SpatialLocationCalculator

**Return** Vector of configuration parameters for ROIs (region of interests)

**struct SpatialLocationCalculatorConfigData**

**struct SpatialLocationCalculatorConfigThresholds**

*#include <RawSpatialLocationCalculatorConfig.hpp>* Spatial location configuration thresholds structure

Contains configuration data for lower and upper threshold in millimeters for ROI. Values outside of threshold range will be ignored when calculating spatial coordinates from depth map.

**class SpatialLocationCalculatorData : public dai::Buffer**

*#include <SpatialLocationCalculatorData.hpp>* *SpatialLocationCalculatorData* message. Carries spatial information (X,Y,Z) and their configuration parameters

## Public Functions

**SpatialLocationCalculatorData ()**

Construct *SpatialLocationCalculatorData* message.

std::vector<*SpatialLocations*> &**getSpatialLocations () const**

Retrieve configuration data for *SpatialLocationCalculatorData*.

**Return** Vector of spatial location data, carrying spatial information (X,Y,Z)

**struct SpatialLocationCalculatorProperties**

*#include <SpatialLocationCalculatorProperties.hpp>* Specify SpatialLocationCalculator options

## Public Members

bool **inputConfigSync** = false  
Whether to wait for config at 'inputConfig' IO.

### struct SpatialLocations

*#include <RawSpatialLocations.hpp>* Spatial location information structure

Contains configuration data, average depth for the calculated ROI on depth map. Together with spatial coordinates: x,y,z. Origin is the center of ROI. Units are in millimeters.

### struct SPIOutProperties

*#include <SPIOutProperties.hpp>* Properties for SPIOut node

## Public Members

std::string **streamName**  
Output stream name.

int **busId**  
SPI bus to use.

### struct StereoDepthProperties

*#include <StereoDepthProperties.hpp>* Specify StereoDepth options

## Public Types

### enum MedianFilter

Median filter config for disparity post-processing

*Values:*

enumerator **MEDIAN\_OFF**

enumerator **KERNEL\_3x3**

enumerator **KERNEL\_5x5**

enumerator **KERNEL\_7x7**

## Public Members

std::vector<std::uint8\_t> **calibration**  
Calibration data byte array

*MedianFilter* **median** = *MedianFilter::KERNEL\_5x5*

Set kernel size for disparity/depth median filtering, or disable

std::int32\_t **confidenceThreshold** = 200  
Confidence threshold for disparity calculation, 0..255

bool **enableLeftRightCheck** = false  
Computes and combines disparities in both L-R and R-L directions, and combine them. For better occlusion handling

bool **enableSubpixel** = false  
Computes disparity with sub-pixel interpolation (5 fractional bits), suitable for long range

```
bool enableExtendedDisparity = false
    Disparity range increased from 96 to 192, combined from full resolution and downscaled images.
    Suitable for short range objects

bool rectifyMirrorFrame = true
    Mirror rectified frames: true to have disparity/depth normal (non-mirrored)

std::int32_t rectifyEdgeFillColor = -1
    Fill color for missing data at frame edges: grayscale 0..255, or -1 to replicate pixels

bool enableOutputRectified = false
    Enable outputting rectified frames. Optimizes computation on device side when disabled

bool enableOutputDepth = false
    Enable outputting 'depth' stream (converted from disparity). In certain configurations, this will dis-
    able 'disparity' stream

tl::optional<std::int32_t> width
    Input frame width. Optional (taken from MonoCamera nodes if they exist)

tl::optional<std::int32_t> height
    Input frame height. Optional (taken from MonoCamera nodes if they exist)

class SystemInformation : public dai::Buffer
    #include <SystemInformation.hpp> SystemInformation message. Carries memory usage, cpu usage and
    chip temperatures.
```

## Public Functions

```
SystemInformation ()
    Construct SystemInformation message.

struct SystemLoggerProperties
    #include <SystemLoggerProperties.hpp> SystemLoggerProperties
```

## Public Members

```
float rateHz = 1.0f
    Rate at which the messages are going to be sent in hertz

struct TensorInfo

struct Timestamp

struct VideoEncoderProperties
    #include <VideoEncoderProperties.hpp> Specify VideoEncoder options such as profile, bitrate, ...
```

## Public Types

```
enum RateControlMode
    Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

    Values:

    enumerator CBR

    enumerator VBR
```

**enum Profile**

Encoding profile, H264, H265 or MJPEG

*Values:*

**enumerator H264\_BASELINE**

**enumerator H264\_HIGH**

**enumerator H264\_MAIN**

**enumerator H265\_MAIN**

**enumerator MJPEG**

**Public Members**

std::int32\_t **bitrate** = 8000

Specifies preferred bitrate (kb) of compressed output bitstream

std::int32\_t **keyframeFrequency** = 30

Every x number of frames a keyframe will be inserted

std::int32\_t **maxBitrate** = 8000

Specifies maximum bitrate (kb) of compressed output bitstream

std::int32\_t **numBframes** = 0

Specifies number of B frames to be inserted

std::uint32\_t **numFramesPool** = 4

This options specifies how many frames are available in this nodes pool (can help if receiver node is slow at consuming)

*Profile* **profile** = *Profile::H264\_BASELINE*

Encoding profile, H264, H265 or MJPEG

std::int32\_t **quality** = 80

Value between 0-100% (approximates quality)

*RateControlMode* **rateCtrlMode** = *RateControlMode::CBR*

Rate control mode specifies if constant or variable bitrate should be used (H264 / H265)

std::int32\_t **width** = 1920

Input and compressed output frame width

std::int32\_t **height** = 1080

Input and compressed output frame height

float **frameRate** = 30.0f

Frame rate

**class XLinkConnection**

*#include <XLinkConnection.hpp>* Represents connection between host and device over XLink protocol

## Public Functions

void **close** ()

Explicitly closes xlink connection.

**Note** This function does not need to be explicitly called as destructor closes the connection automatically

bool **isClosed** () **const**

Is the connection already closed (or disconnected)

**struct XLinkInProperties**

*#include <XLinkInProperties.hpp>* Properties for XLinkIn which define stream name

## Public Members

std::string **streamName**

Name of stream

std::uint32\_t **maxDataSize** = *dai::XLINK\_USB\_BUFFER\_MAX\_SIZE*

Maximum input data size

std::uint32\_t **numFrames** = 8

Number of frames in pool

**struct XLinkOutProperties**

*#include <XLinkOutProperties.hpp>* Properties for XLinkOut which define stream name

## Public Members

float **maxFpsLimit** = -1

Set a limit to how many packets will be sent further to host

std::string **streamName**

Name of stream

bool **metadataOnly** = false

Whether to transfer data or only object attributes

**class XLinkStream**

namespace **bootloader**

## Variables

**constexpr const** char \***XLINK\_CHANNEL\_BOOTLOADER** = "\_\_bootloader"

**constexpr const** char \***XLINK\_CHANNEL\_WATCHDOG** = "\_\_watchdog"

**constexpr** std::uint32\_t **XLINK\_STREAM\_MAX\_SIZE** = 5 \* 1024 \* 1024

**constexpr const** std::chrono::milliseconds **XLINK\_WATCHDOG\_TIMEOUT** = {1500}

namespace **request**



## Enums

enum Command

*Values:*

enumerator USB\_ROM\_BOOT

enumerator BOOT\_APPLICATION

enumerator UPDATE\_FLASH

enumerator GET\_BOOTLOADER\_VERSION

struct BootApplication

struct GetBootloaderVersion

struct UpdateFlash

struct UsbRomBoot

namespace response

## Enums

enum Command

*Values:*

enumerator FLASH\_COMPLETE

enumerator FLASH\_STATUS\_UPDATE

enumerator BOOTLOADER\_VERSION

struct BootloaderVersion

struct FlashComplete

struct FlashStatusUpdate

namespace build

## Variables

constexpr const char \*VERSION = "2.1.0"

constexpr const int VERSION\_MAJOR = 2

constexpr const int VERSION\_MINOR = 1

constexpr const int VERSION\_PATCH = 0

namespace node

class ColorCamera : public dai::Node

*#include <ColorCamera.hpp> ColorCamera* node. For use with color sensors.

## Public Functions

**ColorCamera** (**const** std::shared\_ptr<*PipelineImpl*> &par, int64\_t nodeId)  
Constructs *ColorCamera* node.

void **setBoardSocket** (*CameraBoardSocket* boardSocket)  
Specify which board socket to use

**Parameters**

- boardSocket: Board socket to use

*CameraBoardSocket* **getBoardSocket** () **const**  
Retrieves which board socket to use  
**Return** Board socket to use

void **setCamId** (int64\_t id)  
Set which color camera to use.

int64\_t **getCamId** () **const**  
Get which color camera to use.

void **setImageOrientation** (*CameraImageOrientation* imageOrientation)  
Set camera image orientation.

*CameraImageOrientation* **getImageOrientation** () **const**  
Get camera image orientation.

void **setColorOrder** (*ColorCameraProperties::ColorOrder* colorOrder)  
Set color order of preview output images. RGB or BGR.

*ColorCameraProperties::ColorOrder* **getColorOrder** () **const**  
Get color order of preview output frames. RGB or BGR.

void **setInterleaved** (bool interleaved)  
Set planar or interleaved data of preview output frames.

bool **getInterleaved** () **const**  
Get planar or interleaved data of preview output frames.

void **setFp16** (bool fp16)  
Set fp16 (0..255) data type of preview output frames.

bool **getFp16** () **const**  
Get fp16 (0..255) data of preview output frames.

void **setPreviewSize** (int width, int height)  
Set preview output size.

void **setVideoSize** (int width, int height)  
Set video output size.

void **setStillSize** (int width, int height)  
Set still output size.

void **setResolution** (Properties::SensorResolution resolution)  
Set sensor resolution.

Properties::SensorResolution **getResolution** () **const**  
Get sensor resolution.

void **setFps** (float fps)  
Set rate at which camera should produce frames

**Parameters**

- fps: Rate in frames per second

```

float getFps () const
    Get rate at which camera should produce frames
    Return Rate in frames per second

std::tuple<int, int> getPreviewSize () const
    Get preview size as tuple.

int getPreviewWidth () const
    Get preview width.

int getPreviewHeight () const
    Get preview height.

std::tuple<int, int> getVideoSize () const
    Get video size as tuple.

int getVideoWidth () const
    Get video width.

int getVideoHeight () const
    Get video height.

std::tuple<int, int> getStillSize () const
    Get still size as tuple.

int getStillWidth () const
    Get still width.

int getStillHeight () const
    Get still height.

std::tuple<int, int> getResolutionSize () const
    Get sensor resolution as size.

int getResolutionWidth () const
    Get sensor resolution width.

int getResolutionHeight () const
    Get sensor resolution height.

void sensorCenterCrop ()
    Specify sensor center crop. Resolution size / video size

void setSensorCrop (float x, float y)
    Specifies sensor crop rectangle
    Parameters
    • x: Top left X coordinate
    • y: Top left Y coordinate

std::tuple<float, float> getSensorCrop () const
    Return Sensor top left crop coordinates

float getSensorCropX () const
    Get sensor top left x crop coordinate.

float getSensorCropY () const
    Get sensor top left y crop coordinate.

void setWaitForConfigInput (bool wait)
    Specify to wait until inputConfig receives a configuration message, before sending out a frame.
    Parameters
    • wait: True to wait for inputConfig message, false otherwise

```

bool **getWaitForConfigInput** ()

See [\*setWaitForConfigInput\*](#)

**Return** True if wait for inputConfig message, false otherwise

void **setPreviewKeepAspectRatio** (bool *keep*)

Specifies whether preview output should preserve aspect ratio, after downscaling from video size or not.

**Parameters**

- *keep*: If true, a larger crop region will be considered to still be able to create the final image in the specified aspect ratio. Otherwise video size is resized to fit preview size

bool **getPreviewKeepAspectRatio** ()

See [\*setPreviewKeepAspectRatio\*](#)

**Return** Preview keep aspect ratio option

## Public Members

[\*CameraControl\*](#) **initialControl**

Initial control options to apply to sensor

Input **inputConfig** = { \*this, "inputConfig", Input::Type::SReceiver, false, 8, { { [\*DatatypeEnum::ImageManipConfig\*](#), fa

Input for [\*ImageManipConfig\*](#) message, which can modify crop paremeters in runtime

Default queue is non-blocking with size 8

Input **inputControl** = { \*this, "inputControl", Input::Type::SReceiver, true, 8, { { [\*DatatypeEnum::CameraControl\*](#), fa

Input for [\*CameraControl\*](#) message, which can modify camera parameters in runtime

Default queue is blocking with size 8

Output **video** = { \*this, "video", Output::Type::MSender, { { [\*DatatypeEnum::ImgFrame\*](#), false } } }

Outputs [\*ImgFrame\*](#) message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

Suitable for use with [\*VideoEncoder\*](#) node

Output **preview** = { \*this, "preview", Output::Type::MSender, { { [\*DatatypeEnum::ImgFrame\*](#), false } } }

Outputs [\*ImgFrame\*](#) message that carries BGR/RGB planar/interleaved encoded frame data.

Suitable for use with [\*NeuralNetwork\*](#) node

Output **still** = { \*this, "still", Output::Type::MSender, { { [\*DatatypeEnum::ImgFrame\*](#), false } } }

Outputs [\*ImgFrame\*](#) message that carries NV12 encoded (YUV420, UV plane interleaved) frame data.

The message is sent only when a [\*CameraControl\*](#) message arrives to inputControl with captureStill command set.

**class DetectionNetwork** : public [\*dai::node::NeuralNetwork\*](#)

[\*#include <DetectionNetwork.hpp>\*](#) [\*DetectionNetwork\*](#). Base for different network specializations.

Subclassed by [\*dai::node::MobileNetDetectionNetwork\*](#), [\*dai::node::SpatialDetectionNetwork\*](#), [\*dai::node::YoloDetectionNetwork\*](#)

## Public Functions

void **setConfidenceThreshold** (float *thresh*)

Specifies confidence threshold at which to filter the rest of the detections.

### Parameters

- *thresh*: Detection confidence must be greater than specified threshold to be added to the list

## Public Members

Input **input** = { \*this, "in", Input::Type::SReceiver, true, 5, {{ *DatatypeEnum::Buffer*, true }} }

Input message with data to be inferred upon Default queue is blocking with size 5

Output **out** = { \*this, "out", Output::Type::MSender, {{ *DatatypeEnum::ImgDetections*, false }} }

Outputs *ImgDetections* message that carries parsed detection results.

Output **passthrough** = { \*this, "passthrough", Output::Type::MSender, {{ *DatatypeEnum::Buffer*, true }} }

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

**class ImageManip**: public *dai::Node*

#include <ImageManip.hpp> *ImageManip* node. Capability to crop, resize, warp, ... incoming image frames.

## Public Functions

void **setWaitForConfigInput** (bool *wait*)

Specify whether or not wait until configuration message arrives to inputConfig Input.

### Parameters

- *wait*: True to wait for configuration message, false otherwise

void **setNumFramesPool** (int *numFramesPool*)

Specify number of frames in pool.

### Parameters

- *numFramesPool*: How many frames should the pool have

void **setMaxOutputFrameSize** (int *maxFrameSize*)

Specify maximum size of output image.

### Parameters

- *maxFrameSize*: Maximum frame size in bytes

## Public Members

*ImageManipConfig* **initialConfig**

Initial config to use when manipulating frames

Input **inputConfig** = { \*this, "inputConfig", Input::Type::SReceiver, true, 8, {{ *DatatypeEnum::ImageManipConfig*, true }} }

Input *ImageManipConfig* message with ability to modify parameters in runtime Default queue is blocking with size 8

Input **inputImage** = { \*this, "inputImage", Input::Type::SReceiver, true, 8, {{ *DatatypeEnum::ImgFrame*, true }} }

Input image to be modified Default queue is blocking with size 8

Output **out** = { \*this, "out", Output::Type::MSender, {{ *DatatypeEnum::ImgFrame*, true }} }

Outputs *ImgFrame* message that carries modified image.

```
class MobileNetDetectionNetwork : public dai::node::DetectionNetwork
    #include <DetectionNetwork.hpp> MobileNetDetectionNetwork node. Parses MobileNet results.

class MobileNetSpatialDetectionNetwork : public dai::node::SpatialDetectionNetwork
    #include <SpatialDetectionNetwork.hpp> MobileNetSpatialDetectionNetwork. Mobilenet-SSD
    based network with spatial location data.

class MonoCamera : public dai::Node
    #include <MonoCamera.hpp> MonoCamera node. For use with grayscale sensors.
```

## Public Functions

void **setBoardSocket** (*CameraBoardSocket* boardSocket)

Specify which board socket to use

### Parameters

- boardSocket: Board socket to use

*CameraBoardSocket* **getBoardSocket** () const

Retrieves which board socket to use

**Return** Board socket to use

void **setImageOrientation** (*CameraImageOrientation* imageOrientation)

Set camera image orientation.

*CameraImageOrientation* **getImageOrientation** () const

Get camera image orientation.

void **setResolution** (Properties::SensorResolution resolution)

Set sensor resolution.

Properties::SensorResolution **getResolution** () const

Get sensor resolution.

void **setFps** (float fps)

Set rate at which camera should produce frames

### Parameters

- fps: Rate in frames per second

float **getFps** () const

Get rate at which camera should produce frames

**Return** Rate in frames per second

std::tuple<int, int> **getResolutionSize** () const

Get sensor resolution as size.

int **getResolutionWidth** () const

Get sensor resolution width.

int **getResolutionHeight** () const

Get sensor resolution height.

## Public Members

### *CameraControl* **initialControl**

Initial control options to apply to sensor

Input **inputControl** = { \*this, "inputControl", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::CameraControl*, false } } }

Input for *CameraControl* message, which can modify camera parameters in runtime Default queue is blocking with size 8

Output **out** = { \*this, "out", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) frame data.

Suitable for use *StereoDepth* node

**class** **MyProducer** : **public** *dai::Node*

**class** **NeuralNetwork** : **public** *dai::Node*

#include <NeuralNetwork.hpp> *NeuralNetwork* node. Runs a neural inference on input data.

Subclassed by *dai::node::DetectionNetwork*

## Public Functions

void **setBlobPath** (**const** std::string &path)

Load network blob into assets and use once pipeline is started.

Throws if file doesn't exist or isn't a valid network blob.

### Parameters

- path: Path to network blob

void **setNumPoolFrames** (int numFrames)

Specifies how many frames will be available in the pool

### Parameters

- numFrames: How many frames will pool have

void **setNumInferenceThreads** (int numThreads)

How many threads should the node use to run the network.

### Parameters

- numThreads: Number of threads to dedicate to this node

void **setNumNCEPerInferenceThread** (int numNCEPerThread)

How many Neural Compute Engines should a single thread use for inference

### Parameters

- numNCEPerThread: Number of NCE per thread

int **getNumInferenceThreads** ()

How many inference threads will be used to run the network

**Return** Number of threads, 0, 1 or 2. Zero means AUTO

## Public Members

Input **input** = { \*this, "in", Input::Type::SReceiver, true, 5, { { *DatatypeEnum::Buffer*, true } } }

Input message with data to be inferred upon Default queue is blocking with size 5

Output **out** = { \*this, "out", Output::Type::MSender, { { *DatatypeEnum::NNData*, false } } }

Outputs *NNData* message that carries inference results

Output **passthrough** = { \*this, "passthrough", Output::Type::MSender, { { *DatatypeEnum::Buffer*, true } } }

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.

**class SpatialDetectionNetwork** : public *dai::node::DetectionNetwork*

#include <SpatialDetectionNetwork.hpp> *SpatialDetectionNetwork* node. Runs a neural inference on input image and calculates spatial location data.

Subclassed by *dai::node::MobileNetSpatialDetectionNetwork*, *dai::node::YoloSpatialDetectionNetwork*

## Public Functions

void **setBoundingBoxScaleFactor** (float *scaleFactor*)

Specifies scale factor for detected bounding boxes.

### Parameters

- *scaleFactor*: Scale factor must be in the interval (0,1].

void **setDepthLowerThreshold** (uint32\_t *lowerThreshold*)

Specifies lower threshold in millimeters for depth values which will used to calculate spatial data

### Parameters

- *lowerThreshold*: LowerThreshold must be in the interval [0,upperThreshold] and less than upperThreshold.

void **setDepthUpperThreshold** (uint32\_t *upperThreshold*)

Specifies upper threshold in millimeters for depth values which will used to calculate spatial data

### Parameters

- *upperThreshold*: UpperThreshold must be in the interval (lowerThreshold,65535].

## Public Members

Input **input** = { \*this, "in", Input::Type::SReceiver, true, 5, { { *DatatypeEnum::ImgFrame*, false } } }

Input message with data to be inferred upon Default queue is blocking with size 5

Input **inputDepth** = { \*this, "inputDepth", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgFrame*, false } } }

Input message with depth data used to retrieve spatial information about detected object Default queue is non-blocking with size 4

Output **out** = { \*this, "out", Output::Type::MSender, { { *DatatypeEnum::SpatialImgDetections*, false } } }

Outputs *ImgDetections* message that carries parsed detection results.

Output **boundingBoxMapping** = { \*this, "boundingBoxMapping", Output::Type::MSender, { { *DatatypeEnum::SpatialImgDetections*, false } } }

Outputs mapping of detected bounding boxes relative to depth map

Suitable for when displaying remapped bounding boxes on depth frame

Output **passthrough** = { \*this, "passthrough", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Passthrough message on which the inference was performed.

Suitable for when input queue is set to non-blocking behavior.



Output **passthroughDepth** = { \*this, "passthroughDepth", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }  
 Passthrough message for depth frame on which the spatial location calculation was performed.

Suitable for when input queue is set to non-blocking behavior.

**class SpatialLocationCalculator : public dai::Node**  
*#include <SpatialLocationCalculator.hpp>* *SpatialLocationCalculator* node. Calculates spatial location data on a set of ROIs on depth map.

## Public Functions

void **setWaitForConfigInput** (bool *wait*)  
 Specify whether or not wait until configuration message arrives to inputConfig Input.

### Parameters

- *wait*: True to wait for configuration message, false otherwise.

## Public Members

*SpatialLocationCalculatorConfig* **initialConfig**

Initial config to use when calculating spatial location data.

Input **inputConfig** = { \*this, "inputConfig", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::SpatialLocationCalculatorConfig*, false } } }  
 Input *SpatialLocationCalculatorConfig* message with ability to modify parameters in runtime.  
 Default queue is non-blocking with size 4.

Input **inputDepth** = { \*this, "inputDepth", Input::Type::SReceiver, false, 4, { { *DatatypeEnum::ImgFrame*, false } } }  
 Input message with depth data used to retrieve spatial information about detected object. Default queue is non-blocking with size 4.

Output **out** = { \*this, "out", Output::Type::MSender, { { *DatatypeEnum::SpatialLocationCalculatorData*, false } } }  
 Outputs *SpatialLocationCalculatorData* message that carries spatial location results.

Output **passthroughDepth** = { \*this, "passthroughDepth", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }  
 Passthrough message on which the calculation was performed. Suitable for when input queue is set to non-blocking behavior.

**class SPIOut : public dai::Node**  
*#include <SPIOut.hpp>* *SPIOut* node. Sends messages over SPI.

## Public Functions

void **setStreamName** (std::string *name*)  
 Specifies stream name over which the node will send data

### Parameters

- *name*: Stream name

void **setBusId** (int *id*)  
 Specifies SPI Bus number to use

### Parameters

- *id*: SPI Bus id

## Public Members

Input **input** = { \*this, "in", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::Buffer*, true } } }

Input for any type of messages to be transferred over SPI stream

Default queue is blocking with size 8

**class StereoDepth** : public *dai::Node*  
*#include <StereoDepth.hpp>* *StereoDepth* node. Compute stereo disparity and depth from left-right image pair.

## Public Functions

void **loadCalibrationFile** (const std::string &path)  
Specify local filesystem path to the calibration file

### Parameters

- path: Path to calibration file. If empty use EEPROM

void **loadCalibrationData** (const std::vector<std::uint8\_t> &data)  
Specify calibration data as a vector of bytes

### Parameters

- path: Calibration data. If empty use EEPROM

void **setEmptyCalibration** ()  
Specify that a passthrough/dummy calibration should be used, when input frames are already rectified (e.g. sourced from recordings on the host)

void **setInputResolution** (int width, int height)  
Specify input resolution size

Optional if *MonoCamera* exists, otherwise necessary

void **setMedianFilter** (Properties::MedianFilter median)

### Parameters

- median: Set kernel size for disparity/depth median filtering, or disable

void **setConfidenceThreshold** (int confThr)  
Confidence threshold for disparity calculation

### Parameters

- confThr: Confidence threshold value 0..255

void **setLeftRightCheck** (bool enable)  
Computes and combines disparities in both L-R and R-L directions, and combine them.

For better occlusion handling

void **setSubpixel** (bool enable)  
Computes disparity with sub-pixel interpolation (5 fractional bits).

Suitable for long range

void **setExtendedDisparity** (bool enable)  
Disparity range increased from 96 to 192, combined from full resolution and downscaled images.

Suitable for short range objects

void **setRectifyEdgeFillColor** (int color)  
Fill color for missing data at frame edges

### Parameters

- color: Grayscale 0..255, or -1 to replicate pixels

void **setRectifyMirrorFrame** (bool *enable*)

Mirror rectified frames

**Parameters**

- *enable*: True for normal disparity/depth, otherwise mirrored

void **setOutputRectified** (bool *enable*)

Enable outputting rectified frames. Optimizes computation on device side when disabled

void **setOutputDepth** (bool *enable*)

Enable outputting 'depth' stream (converted from disparity). In certain configurations, this will disable 'disparity' stream

## Public Members

Input **left** = { \*this, "left", Input::Type::SReceiver, false, 8, { { *DatatypeEnum::ImgFrame*, true } } }

Input for left *ImgFrame* of left-right pair

Default queue is non-blocking with size 8

Input **right** = { \*this, "right", Input::Type::SReceiver, false, 8, { { *DatatypeEnum::ImgFrame*, true } } }

Input for right *ImgFrame* of left-right pair

Default queue is non-blocking with size 8

Output **depth** = { \*this, "depth", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW16 encoded (0..65535) depth data in millimeters.

Output **disparity** = { \*this, "disparity", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW8 encoded (0..96 or 0..192 for Extended mode) disparity data.

Output **syncedLeft** = { \*this, "syncedLeft", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Passthrough *ImgFrame* message from 'left' Input.

Output **syncedRight** = { \*this, "syncedRight", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Passthrough *ImgFrame* message from 'right' Input.

Output **rectifiedLeft** = { \*this, "rectifiedLeft", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) rectified frame data.

Output **rectifiedRight** = { \*this, "rectifiedRight", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries RAW8 encoded (grayscale) rectified frame data.

**class SystemLogger**: public *dai::Node*

#include <SystemLogger.hpp> *SystemLogger* node. Send system information periodically.

## Public Functions

void **setRate** (float *hz*)

Specify logging rate, at which messages will be sent to out output

**Parameters**

- *hz*: Sending rate in hertz (messages per second)

## Public Members

Output **out** = {\*this, "out", Output::Type::MSender, {{*DatatypeEnum::SystemInformation*, false}}} Outputs *SystemInformation* message that carries various system information like memory and CPU usage, temperatures, ...

**class VideoEncoder** : public *dai::Node*  
#include <VideoEncoder.hpp> *VideoEncoder* node. Encodes frames into MJPEG, H264 or H265.

## Public Functions

void **setDefaultProfilePreset** (int *width*, int *height*, float *fps*, Properties::Profile *profile*)

Sets a default preset based on specified input size, frame rate and profile

### Parameters

- *width*: Input frame width
- *height*: Input frame height
- *fps*: Frame rate in frames per second
- *profile*: Encoding profile

void **setDefaultProfilePreset** (std::tuple<int, int> *size*, float *fps*, Properties::Profile *profile*)

Sets a default preset based on specified input size, frame rate and profile

### Parameters

- *size*: Input frame size
- *fps*: Frame rate in frames per second
- *profile*: Encoding profile

void **setNumFramesPool** (int *frames*)

Set number of frames in pool

### Parameters

- *frames*: Number of pool frames

int **getNumFramesPool** () const

Get number of frames in pool

**Return** Number of pool frames

void **setRateControlMode** (Properties::RateControlMode *mode*)

Set rate control mode.

void **setProfile** (int *width*, int *height*, Properties::Profile *profile*)

Set encoding profile.

void **setBitrate** (int *bitrate*)

Set output bitrate in bps. Final bitrate depends on rate control mode.

void **setBitrateKbps** (int *bitrateKbps*)

Set output bitrate in kbps. Final bitrate depends on rate control mode.

void **setKeyframeFrequency** (int *freq*)

Set keyframe frequency. Every Nth frame a keyframe is inserted.

Applicable only to H264 and H265 profiles

Examples:

- 30 FPS video, keyframe frequency: 30. Every 1s a keyframe will be inserted
- 60 FPS video, keyframe frequency: 180. Every 3s a keyframe will be inserted

```

void setNumBFrames (int numBFrames)
    Set number of B frames to be inserted.

void setQuality (int quality)
    Set quality
    Parameters
    • quality: Value between 0-100%. Approximates quality

void setFrameRate (int frameRate)
    Sets expected frame rate
    Parameters
    • frameRate: Frame rate in frames per second

Properties::RateControlMode getRateControlMode () const
    Get rate control mode.

Properties::Profile getProfile () const
    Get profile.

int getBitrate () const
    Get bitrate in bps.

int getBitrateKbps () const
    Get bitrate in kbps.

int getKeyframeFrequency () const
    Get keyframe frequency.

int getNumBFrames () const
    Get number of B frames.

int getQuality () const
    Get quality.

std::tuple<int, int> getSize () const
    Get input size.

int getWidth () const
    Get input width.

int getHeight () const
    Get input height.

int getFrameRate () const
    Get frame rate.

```

### Public Members

Input **input** = { \*this, "in", Input::Type::SReceiver, true, 4, { { *DatatypeEnum::ImgFrame*, true } } }

Input for NV12 *ImgFrame* to be encoded Default queue is blocking with size set by 'setNumFramesPool' (4).

Output **bitstream** = { \*this, "bitstream", Output::Type::MSender, { { *DatatypeEnum::ImgFrame*, false } } }

Outputs *ImgFrame* message that carries BITSTREAM encoded (MJPEG, H264 or H265) frame data.

```

class XLinkIn: public dai::Node
    #include <XLinkIn.hpp> XLinkIn node. Receives messages over XLink.

```

## Public Functions

void **setStreamName** (const std::string &name)

Specifies XLink stream name to use.

The name should not start with double underscores ‘\_\_’, as those are reserved for internal use.

### Parameters

- name: Stream name

void **setMaxDataSize** (std::uint32\_t maxSize)

Set maximum message size it can receive

### Parameters

- maxSize: Maximum size in bytes

void **setNumFrames** (std::uint32\_t numFrames)

Set number of frames in pool for sending messages forward

### Parameters

- numFrames: Maximum number of frames in pool

std::string **getStreamName** () const

Get stream name.

std::uint32\_t **getMaxDataSize** () const

Get maximum messages size in bytes.

std::uint32\_t **getNumFrames** () const

Get number of frames in pool.

## Public Members

Output **out** = { \*this, "out", Output::Type::MSender, { { *DatatypeEnum::Buffer*, true } } }

Outputs message of same type as send from host.

**class XLinkOut** : public *dai::Node*

*#include <XLinkOut.hpp>* *XLinkOut* node. Sends messages over XLink.

## Public Functions

void **setStreamName** (const std::string &name)

Specifies XLink stream name to use.

The name should not start with double underscores ‘\_\_’, as those are reserved for internal use.

### Parameters

- name: Stream name

void **setFpsLimit** (float fps)

Specifies a message sending limit. It's approximated from specified rate.

### Parameters

- fps: Approximate rate limit in messages per second

void **setMetadataOnly** (bool metadataOnly)

Specify whether to transfer only messages attributes and not buffer data

std::string **getStreamName** () const

Get stream name.

float **getFpsLimit** () const

Get rate limit in messages per second.

bool **getMetadataOnly** () const

Get whether to transfer only messages attributes and not buffer data.

### Public Members

Input **input** = { \*this, "in", Input::Type::SReceiver, true, 8, { { *DatatypeEnum::Buffer*, true } } }

Input for any type of messages to be transfered over XLink stream

Default queue is blocking with size 8

**class YoloDetectionNetwork** : public *dai::node::DetectionNetwork*  
*#include <DetectionNetwork.hpp>* *YoloDetectionNetwork* node. Parses Yolo results.

### Public Functions

void **setNumClasses** (const int *numClasses*)

Set num classes.

void **setCoordinateSize** (const int *coordinates*)

Set coordianate size.

void **setAnchors** (std::vector<float> *anchors*)

Set anchors.

void **setAnchorMasks** (std::map<std::string, std::vector<int>> *anchorMasks*)

Set anchor masks.

void **setIouThreshold** (float *thresh*)

Set Iou threshold.

**class YoloSpatialDetectionNetwork** : public *dai::node::SpatialDetectionNetwork*  
*#include <SpatialDetectionNetwork.hpp>* *YoloSpatialDetectionNetwork*. (tiny)Yolov3/v4 based network with spatial location data.

### Public Functions

void **setNumClasses** (const int *numClasses*)

Set num classes.

void **setCoordinateSize** (const int *coordinates*)

Set coordianate size.

void **setAnchors** (std::vector<float> *anchors*)

Set anchors.

void **setAnchorMasks** (std::map<std::string, std::vector<int>> *anchorMasks*)

Set anchor masks.

void **setIouThreshold** (float *thresh*)

Set Iou threshold.

We're always happy to help with code or other questions you might have.





## PYTHON MODULE INDEX

### d

`depthai`, [99](#)



## Symbols

- `__init__()` (*depthai.ADatatype* method), 101
- `__init__()` (*depthai.Asset* method), 101
- `__init__()` (*depthai.AssetManager* method), 102
- `__init__()` (*depthai.Buffer* method), 103
- `__init__()` (*depthai.CameraBoardSocket* method), 104
- `__init__()` (*depthai.CameraControl* method), 110
- `__init__()` (*depthai.CameraControl.AntiBandingMode* method), 106
- `__init__()` (*depthai.CameraControl.AutoFocusMode* method), 106
- `__init__()` (*depthai.CameraControl.AutoWhiteBalanceMode* method), 107
- `__init__()` (*depthai.CameraControl.EffectMode* method), 108
- `__init__()` (*depthai.CameraControl.SceneMode* method), 110
- `__init__()` (*depthai.CameraImageOrientation* method), 113
- `__init__()` (*depthai.ChipTemperature* method), 113
- `__init__()` (*depthai.ColorCamera* method), 116
- `__init__()` (*depthai.ColorCameraProperties* method), 120
- `__init__()` (*depthai.ColorCameraProperties.ColorOrder* method), 119
- `__init__()` (*depthai.ColorCameraProperties.SensorResolution* method), 120
- `__init__()` (*depthai.CpuUsage* method), 121
- `__init__()` (*depthai.DataInputQueue* method), 121
- `__init__()` (*depthai.DataOutputQueue* method), 122
- `__init__()` (*depthai.DetectionNetwork* method), 124
- `__init__()` (*depthai.DetectionNetworkProperties* method), 125
- `__init__()` (*depthai.Device* method), 126
- `__init__()` (*depthai.DeviceBootloader* method), 133
- `__init__()` (*depthai.DeviceBootloader.Version* method), 133
- `__init__()` (*depthai.DeviceDesc* method), 135
- `__init__()` (*depthai.DeviceInfo* method), 135
- `__init__()` (*depthai.GlobalProperties* method), 136
- `__init__()` (*depthai.ImageManip* method), 136
- `__init__()` (*depthai.ImageManipConfig* method), 138
- `__init__()` (*depthai.ImageDetection* method), 141
- `__init__()` (*depthai.ImageDetections* method), 141
- `__init__()` (*depthai.ImageFrame* method), 145
- `__init__()` (*depthai.ImageFrame.Specs* method), 142
- `__init__()` (*depthai.ImageFrame.Type* method), 145
- `__init__()` (*depthai.LogLevel* method), 147
- `__init__()` (*depthai.MemoryInfo* method), 147
- `__init__()` (*depthai.MobileNetDetectionNetwork* method), 148
- `__init__()` (*depthai.MobileNetSpatialDetectionNetwork* method), 148
- `__init__()` (*depthai.MonoCamera* method), 149
- `__init__()` (*depthai.MonoCameraProperties* method), 151
- `__init__()` (*depthai.MonoCameraProperties.SensorResolution* method), 151
- `__init__()` (*depthai.NNData* method), 152
- `__init__()` (*depthai.NeuralNetwork* method), 154
- `__init__()` (*depthai.NeuralNetworkProperties* method), 155
- `__init__()` (*depthai.Node* method), 158
- `__init__()` (*depthai.Node.Connection* method), 156
- `__init__()` (*depthai.Node.Id* method), 156
- `__init__()` (*depthai.Node.Input* method), 157
- `__init__()` (*depthai.Node.Output* method), 157
- `__init__()` (*depthai.OpenVINO* method), 160
- `__init__()` (*depthai.OpenVINO.Version* method), 159
- `__init__()` (*depthai.Pipeline* method), 161
- `__init__()` (*depthai.Point2f* method), 163
- `__init__()` (*depthai.Point3f* method), 163
- `__init__()` (*depthai.RawBuffer* method), 164
- `__init__()` (*depthai.RawCameraControl* method), 169
- `__init__()` (*depthai.RawCameraControl.AntiBandingMode* method), 165
- `__init__()` (*depthai.RawCameraControl.AutoFocusMode* method), 166
- `__init__()` (*depthai.RawCameraControl.AutoWhiteBalanceMode* method), 167
- `__init__()` (*depthai.RawCameraControl.EffectMode* method), 168

`__init__()` (*depthai.RawCameraControl.SceneMode* method), 169  
`__init__()` (*depthai.RawImageManipConfig* method), 172  
`__init__()` (*depthai.RawImageManipConfig.CropConfig* method), 170  
`__init__()` (*depthai.RawImageManipConfig.CropRect* method), 171  
`__init__()` (*depthai.RawImageManipConfig.FormatConfig* method), 171  
`__init__()` (*depthai.RawImageManipConfig.ResizeConfig* method), 172  
`__init__()` (*depthai.RawImgDetections* method), 173  
`__init__()` (*depthai.RawImgFrame* method), 177  
`__init__()` (*depthai.RawImgFrame.Specs* method), 173  
`__init__()` (*depthai.RawImgFrame.Type* method), 176  
`__init__()` (*depthai.RawNNData* method), 177  
`__init__()` (*depthai.RawSpatialImgDetections* method), 177  
`__init__()` (*depthai.RawSystemInformation* method), 178  
`__init__()` (*depthai.Rect* method), 178  
`__init__()` (*depthai.RotatedRect* method), 179  
`__init__()` (*depthai.SPIOut* method), 180  
`__init__()` (*depthai.Size2f* method), 180  
`__init__()` (*depthai.SpatialDetectionNetwork* method), 181  
`__init__()` (*depthai.SpatialDetectionNetworkProperties* method), 182  
`__init__()` (*depthai.SpatialImgDetection* method), 183  
`__init__()` (*depthai.SpatialImgDetections* method), 183  
`__init__()` (*depthai.SpatialLocationCalculator* method), 184  
`__init__()` (*depthai.SpatialLocationCalculatorConfig* method), 185  
`__init__()` (*depthai.SpatialLocationCalculatorConfigData* method), 185  
`__init__()` (*depthai.SpatialLocationCalculatorConfigThresholds* method), 186  
`__init__()` (*depthai.SpatialLocationCalculatorData* method), 186  
`__init__()` (*depthai.SpatialLocationCalculatorProperties* method), 186  
`__init__()` (*depthai.SpatialLocations* method), 187  
`__init__()` (*depthai.StereoDepth* method), 188  
`__init__()` (*depthai.StereoDepthProperties* method), 191  
`__init__()` (*depthai.StereoDepthProperties.MedianFilter* method), 191  
`__init__()` (*depthai.SystemInformation* method), 192  
`__init__()` (*depthai.SystemLogger* method), 192  
`__init__()` (*depthai.SystemLoggerProperties* method), 193  
`__init__()` (*depthai.TensorInfo* method), 195  
`__init__()` (*depthai.TensorInfo.DataType* method), 194  
`__init__()` (*depthai.TensorInfo.StorageOrder* method), 195  
`__init__()` (*depthai.Timestamp* method), 196  
`__init__()` (*depthai.VideoEncoder* method), 197  
`__init__()` (*depthai.VideoEncoderProperties* method), 201  
`__init__()` (*depthai.VideoEncoderProperties.Profile* method), 200  
`__init__()` (*depthai.VideoEncoderProperties.RateControlMode* method), 201  
`__init__()` (*depthai.XLinkConnection* method), 201  
`__init__()` (*depthai.XLinkDeviceState* method), 202  
`__init__()` (*depthai.XLinkIn* method), 203  
`__init__()` (*depthai.XLinkOut* method), 204  
`__init__()` (*depthai.XLinkPlatform* method), 205  
`__init__()` (*depthai.XLinkProtocol* method), 205  
`__init__()` (*depthai.YoloDetectionNetwork* method), 206  
`__init__()` (*depthai.YoloSpatialDetectionNetwork* method), 206

## A

**ACTION** (*depthai.CameraControl.SceneMode* attribute), 110  
**ACTION** (*depthai.RawCameraControl.SceneMode* attribute), 169  
**ADatatype** (class in *depthai*), 100  
**add()** (*depthai.AssetManager* method), 102  
**addCallback()** (*depthai.DataOutputQueue* method), 122  
**addExisting()** (*depthai.AssetManager* method), 102  
**addLogCallback()** (*depthai.Device* method), 127  
**addROI()** (*depthai.SpatialLocationCalculatorConfig* method), 185  
**alignment()** (*depthai.Asset* property), 101  
**anchorsMasks()** (*depthai.DetectionNetworkProperties* property), 125  
**anchors()** (*depthai.DetectionNetworkProperties* property), 125  
**angle()** (*depthai.RotatedRect* property), 180  
**AQUA** (*depthai.CameraControl.EffectMode* attribute), 108  
**AQUA** (*depthai.RawCameraControl.EffectMode* attribute), 167  
**area()** (*depthai.Rect* method), 179  
**areVersionsBlobCompatible()** (*depthai.OpenVINO* static method), 160  
**Asset** (class in *depthai*), 101

- AssetManager (class in *depthai*), 101
- AUTO (*depthai.CameraBoardSocket* attribute), 104
- AUTO (*depthai.CameraControl.AntiBandingMode* attribute), 105
- AUTO (*depthai.CameraControl.AutoFocusMode* attribute), 106
- AUTO (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 107
- AUTO (*depthai.CameraImageOrientation* attribute), 113
- AUTO (*depthai.RawCameraControl.AntiBandingMode* attribute), 165
- AUTO (*depthai.RawCameraControl.AutoFocusMode* attribute), 165
- AUTO (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 166
- autoFocusMode () (*depthai.RawCameraControl* property), 169
- average () (*depthai.ChipTemperature* property), 113
- average () (*depthai.CpuUsage* property), 121
- ## B
- BARCODE (*depthai.CameraControl.SceneMode* attribute), 110
- BARCODE (*depthai.RawCameraControl.SceneMode* attribute), 169
- batchSize () (*depthai.RawNNData* property), 177
- BEACH (*depthai.CameraControl.SceneMode* attribute), 110
- BEACH (*depthai.RawCameraControl.SceneMode* attribute), 169
- bgBlue () (*depthai.RawImageManipConfig.ResizeConfig* property), 172
- bgGreen () (*depthai.RawImageManipConfig.ResizeConfig* property), 172
- BGR (*depthai.ColorCameraProperties.ColorOrder* attribute), 119
- BGR888i (*depthai.ImgFrame.Type* attribute), 144
- BGR888i (*depthai.RawImgFrame.Type* attribute), 176
- BGR888p (*depthai.ImgFrame.Type* attribute), 144
- BGR888p (*depthai.RawImgFrame.Type* attribute), 176
- bgRed () (*depthai.RawImageManipConfig.ResizeConfig* property), 172
- BGRF16F16F16i (*depthai.ImgFrame.Type* attribute), 144
- BGRF16F16F16i (*depthai.RawImgFrame.Type* attribute), 176
- BGRF16F16F16p (*depthai.ImgFrame.Type* attribute), 144
- BGRF16F16F16p (*depthai.RawImgFrame.Type* attribute), 176
- bitrate () (*depthai.VideoEncoderProperties* property), 201
- BITSTREAM (*depthai.ImgFrame.Type* attribute), 144
- BITSTREAM (*depthai.RawImgFrame.Type* attribute), 176
- bitstream () (*depthai.VideoEncoder* property), 197
- BLACKBOARD (*depthai.CameraControl.EffectMode* attribute), 108
- BLACKBOARD (*depthai.RawCameraControl.EffectMode* attribute), 167
- blobSize () (*depthai.NeuralNetworkProperties* property), 155
- blobUri () (*depthai.NeuralNetworkProperties* property), 155
- boardSocket () (*depthai.ColorCameraProperties* property), 120
- boardSocket () (*depthai.MonoCameraProperties* property), 151
- bottomRight () (*depthai.Rect* method), 179
- boundingBoxMapping () (*depthai.SpatialDetectionNetwork* property), 182
- Buffer (class in *depthai*), 103
- bytesPP () (*depthai.ImgFrame.Specs* property), 142
- bytesPP () (*depthai.RawImgFrame.Specs* property), 173
- ## C
- C (*depthai.TensorInfo.StorageOrder* attribute), 195
- calibration () (*depthai.StereoDepthProperties* property), 191
- CameraBoardSocket (class in *depthai*), 103
- CameraControl (class in *depthai*), 104
- CameraControl.AntiBandingMode (class in *depthai*), 105
- CameraControl.AutoFocusMode (class in *depthai*), 106
- CameraControl.AutoWhiteBalanceMode (class in *depthai*), 106
- CameraControl.EffectMode (class in *depthai*), 107
- CameraControl.SceneMode (class in *depthai*), 109
- CameraImageOrientation (class in *depthai*), 112
- canConnect () (*depthai.Node.Output* method), 157
- CANDLELIGHT (*depthai.CameraControl.SceneMode* attribute), 110
- CANDLELIGHT (*depthai.RawCameraControl.SceneMode* attribute), 169
- category () (*depthai.RawImgFrame* property), 177
- CBR (*depthai.VideoEncoderProperties.RateControlMode* attribute), 201
- center () (*depthai.RotatedRect* property), 180
- ChipTemperature (class in *depthai*), 113
- chipTemperature () (*depthai.RawSystemInformation* property), 178

chipTemperature() (*depthai.SystemInformation property*), 192  
 CHW (*depthai.TensorInfo.StorageOrder attribute*), 195  
 classes() (*depthai.DetectionNetworkProperties property*), 125  
 close() (*depthai.Device method*), 127  
 close() (*depthai.DeviceBootloader method*), 133  
 CLOUDY\_DAYLIGHT (*depthai.CameraControl.AutoFocusMode attribute*), 107  
 CLOUDY\_DAYLIGHT (*depthai.RawCameraControl.AutoFocusMode attribute*), 166  
 cmdMask() (*depthai.RawCameraControl property*), 169  
 cmxMemoryUsage() (*depthai.RawSystemInformation property*), 178  
 cmxMemoryUsage() (*depthai.SystemInformation property*), 192  
 CN (*depthai.TensorInfo.StorageOrder attribute*), 195  
 ColorCamera (class in *depthai*), 113  
 ColorCameraProperties (class in *depthai*), 118  
 ColorCameraProperties.ColorOrder (class in *depthai*), 119  
 ColorCameraProperties.SensorResolution (class in *depthai*), 120  
 colorOrder() (*depthai.ColorCameraProperties property*), 120  
 confidence() (*depthai.ImgDetection property*), 141  
 confidenceThreshold() (*depthai.DetectionNetworkProperties property*), 125  
 confidenceThreshold() (*depthai.StereoDepthProperties property*), 191  
 config() (*depthai.SpatialLocations property*), 187  
 contains() (*depthai.Rect method*), 179  
 CONTINUOUS\_PICTURE (*depthai.CameraControl.AutoFocusMode attribute*), 106  
 CONTINUOUS\_PICTURE (*depthai.RawCameraControl.AutoFocusMode attribute*), 165  
 CONTINUOUS\_VIDEO (*depthai.CameraControl.AutoFocusMode attribute*), 106  
 CONTINUOUS\_VIDEO (*depthai.RawCameraControl.AutoFocusMode attribute*), 166  
 coordinates() (*depthai.DetectionNetworkProperties property*), 125  
 CpuUsage (class in *depthai*), 121  
 createColorCamera() (*depthai.Pipeline method*), 161  
 createDepthaiApplicationPackage() (*depthai.DeviceBootloader static method*), 133  
 createImageManip() (*depthai.Pipeline method*), 161  
 createMobileNetDetectionNetwork() (*depthai.Pipeline method*), 161  
 createMobileNetSpatialDetectionNetwork() (*depthai.Pipeline method*), 161  
 createMonoCamera() (*depthai.Pipeline method*), 161  
 createNaturalNetwork() (*depthai.Pipeline method*), 161  
 createNaturalLocationCalculator() (*depthai.Pipeline method*), 161  
 createSPIOut() (*depthai.Pipeline method*), 161  
 createStereoDepth() (*depthai.Pipeline method*), 161  
 createSystemLogger() (*depthai.Pipeline method*), 161  
 createVideoEncoder() (*depthai.Pipeline method*), 161  
 createXLinkIn() (*depthai.Pipeline method*), 161  
 createXLinkOut() (*depthai.Pipeline method*), 161  
 createYoloDetectionNetwork() (*depthai.Pipeline method*), 161  
 createYoloSpatialDetectionNetwork() (*depthai.Pipeline method*), 162  
 CRITICAL (*depthai.LogLevel attribute*), 147  
 cropConfig() (*depthai.RawImageManipConfig property*), 172  
 cropRatio() (*depthai.RawImageManipConfig.CropConfig property*), 170  
 cropRect() (*depthai.RawImageManipConfig.CropConfig property*), 170  
 cropRotatedRect() (*depthai.RawImageManipConfig.CropConfig property*), 170  
 css() (*depthai.ChipTemperature property*), 113  
 CWH (*depthai.TensorInfo.StorageOrder attribute*), 195

## D

dai (C++ type), 207  
 dai::ADatatype (C++ class), 208  
 dai::Asset (C++ struct), 208  
 dai::AssetManager (C++ class), 208  
 dai::AssetManager::add (C++ function), 209  
 dai::AssetManager::addExisting (C++ function), 209  
 dai::AssetManager::get (C++ function), 209  
 dai::AssetManager::getAll (C++ function), 209  
 dai::AssetManager::remove (C++ function), 209  
 dai::AssetManager::serialize (C++ function), 209  
 dai::AssetManager::set (C++ function), 209  
 dai::AssetManager::size (C++ function), 209



dai::Assets (C++ class), 209  
 dai::AssetsMutable (C++ class), 209  
 dai::AssetView (C++ struct), 209  
 dai::bootloader (C++ type), 244  
 dai::bootloader::request (C++ type), 244  
 dai::bootloader::request::BootApplication (C++ struct), 245  
 dai::bootloader::request::Command (C++ enum), 245  
 dai::bootloader::request::Command::BOOT\_APPLICATION (C++ enumerator), 245  
 dai::bootloader::request::Command::GET\_BOOTLOADER\_VERSION (C++ enumerator), 245  
 dai::bootloader::request::Command::UPDATE\_FLASH (C++ enumerator), 245  
 dai::bootloader::request::Command::USB\_ROM\_BOOT (C++ enumerator), 245  
 dai::bootloader::request::GetBootloaderVersion (C++ struct), 245  
 dai::bootloader::request::UpdateFlash (C++ struct), 245  
 dai::bootloader::request::UsbRomBoot (C++ struct), 245  
 dai::bootloader::response (C++ type), 245  
 dai::bootloader::response::BootloaderVersion (C++ struct), 245  
 dai::bootloader::response::Command (C++ enum), 245  
 dai::bootloader::response::Command::BOOTLOADER\_VERSION (C++ enumerator), 245  
 dai::bootloader::response::Command::FLASH\_COMPILE (C++ enumerator), 245  
 dai::bootloader::response::Command::FLASH\_STATUS\_UPDATE (C++ enumerator), 245  
 dai::bootloader::response::FlashComplete (C++ struct), 245  
 dai::bootloader::response::FlashStatusUpdate (C++ struct), 245  
 dai::bootloader::XLINK\_CHANNEL\_BOOTLOADER (C++ member), 244  
 dai::bootloader::XLINK\_CHANNEL\_WATCHDOG (C++ member), 244  
 dai::bootloader::XLINK\_STREAM\_MAX\_SIZE (C++ member), 244  
 dai::bootloader::XLINK\_WATCHDOG\_TIMEOUT (C++ member), 244  
 dai::Buffer (C++ class), 209  
 dai::Buffer::Buffer (C++ function), 210  
 dai::Buffer::getData (C++ function), 210  
 dai::Buffer::setData (C++ function), 210  
 dai::build (C++ type), 245  
 dai::build::VERSION (C++ member), 245  
 dai::build::VERSION\_MAJOR (C++ member), 245  
 dai::build::VERSION\_MINOR (C++ member), 245  
 dai::build::VERSION\_PATCH (C++ member), 245  
 dai::CallbackHandler (C++ class), 210  
 dai::CameraBoardSocket (C++ enum), 207  
 dai::CameraBoardSocket::AUTO (C++ enumerator), 207  
 dai::CameraBoardSocket::LEFT (C++ enumerator), 207  
 dai::CameraBoardSocket::RGB (C++ enumerator), 207  
 dai::CameraBoardSocket::RIGHT (C++ enumerator), 207  
 dai::CameraControl (C++ class), 210  
 dai::CameraControl::CameraControl (C++ function), 210  
 dai::CameraControl::getCaptureStill (C++ function), 212  
 dai::CameraControl::setAntiBandingMode (C++ function), 211  
 dai::CameraControl::setAutoExposureCompensation (C++ function), 211  
 dai::CameraControl::setAutoExposureEnable (C++ function), 211  
 dai::CameraControl::setAutoExposureLock (C++ function), 211  
 dai::CameraControl::setAutoExposureRegion (C++ function), 211  
 dai::CameraControl::setAutoFocusMode (C++ function), 210  
 dai::CameraControl::setAutoFocusRegion (C++ function), 210  
 dai::CameraControl::setAutoFocusTrigger (C++ function), 210  
 dai::CameraControl::setAutoWhiteBalanceLock (C++ function), 211  
 dai::CameraControl::setAutoWhiteBalanceMode (C++ function), 211  
 dai::CameraControl::setBrightness (C++ function), 211  
 dai::CameraControl::setCaptureStill (C++ function), 210  
 dai::CameraControl::setChromaDenoise (C++ function), 212  
 dai::CameraControl::setContrast (C++ function), 211  
 dai::CameraControl::setEffectMode (C++ function), 212  
 dai::CameraControl::setLumaDenoise (C++ function), 212  
 dai::CameraControl::setManualExposure (C++ function), 211  
 dai::CameraControl::setManualFocus (C++

*function*), 211  
 dai::CameraControl::setNoiseReductionStrength (C++ *function*), 212  
 dai::CameraControl::setSaturation (C++ *function*), 212  
 dai::CameraControl::setSceneMode (C++ *function*), 212  
 dai::CameraControl::setSharpness (C++ *function*), 212  
 dai::CameraControl::setStartStreaming (C++ *function*), 210  
 dai::CameraControl::setStopStreaming (C++ *function*), 210  
 dai::CameraImageOrientation (C++ *enum*), 207  
 dai::CameraImageOrientation::AUTO (C++ *enumerator*), 207  
 dai::CameraImageOrientation::HORIZONTAL\_MIRROR (C++ *enumerator*), 207  
 dai::CameraImageOrientation::NORMAL (C++ *enumerator*), 207  
 dai::CameraImageOrientation::ROTATE\_180\_DEG (C++ *enumerator*), 207  
 dai::CameraImageOrientation::VERTICAL\_FLIP (C++ *enumerator*), 207  
 dai::ChipTemperature (C++ *struct*), 212  
 dai::ChipTemperature::average (C++ *member*), 212  
 dai::ChipTemperature::css (C++ *member*), 212  
 dai::ChipTemperature::dss (C++ *member*), 212  
 dai::ChipTemperature::mss (C++ *member*), 212  
 dai::ChipTemperature::upa (C++ *member*), 212  
 dai::ColorCameraProperties (C++ *struct*), 213  
 dai::ColorCameraProperties::boardSocket (C++ *member*), 213  
 dai::ColorCameraProperties::ColorOrder (C++ *enum*), 213  
 dai::ColorCameraProperties::colorOrder (C++ *member*), 213  
 dai::ColorCameraProperties::ColorOrder::RGB (C++ *enumerator*), 213  
 dai::ColorCameraProperties::ColorOrder::BGR (C++ *enumerator*), 213  
 dai::ColorCameraProperties::fp16 (C++ *member*), 213  
 dai::ColorCameraProperties::fps (C++ *member*), 214  
 dai::ColorCameraProperties::imageOrientation (C++ *member*), 213  
 dai::ColorCameraProperties::inputConfigSync (C++ *member*), 214  
 dai::ColorCameraProperties::interleaved (C++ *member*), 213  
 dai::ColorCameraProperties::previewHeight (C++ *member*), 213  
 dai::ColorCameraProperties::previewKeepAspectRatio (C++ *member*), 214  
 dai::ColorCameraProperties::previewWidth (C++ *member*), 213  
 dai::ColorCameraProperties::resolution (C++ *member*), 213  
 dai::ColorCameraProperties::sensorCropX (C++ *member*), 214  
 dai::ColorCameraProperties::SensorResolution (C++ *enum*), 213  
 dai::ColorCameraProperties::SensorResolution::THE\_1 (C++ *enumerator*), 213  
 dai::ColorCameraProperties::SensorResolution::THE\_2 (C++ *enumerator*), 213  
 dai::ColorCameraProperties::SensorResolution::THE\_4 (C++ *enumerator*), 213  
 dai::ColorCameraProperties::stillHeight (C++ *member*), 213  
 dai::ColorCameraProperties::stillWidth (C++ *member*), 213  
 dai::ColorCameraProperties::videoHeight (C++ *member*), 213  
 dai::ColorCameraProperties::videoWidth (C++ *member*), 213  
 dai::CpuUsage (C++ *struct*), 214  
 dai::CpuUsage::average (C++ *member*), 214  
 dai::CpuUsage::msTime (C++ *member*), 214  
 dai::DataInputQueue (C++ *class*), 214  
 dai::DataInputQueue::getBlocking (C++ *function*), 214  
 dai::DataInputQueue::getMaxDataSize (C++ *function*), 214  
 dai::DataInputQueue::getMaxSize (C++ *function*), 215  
 dai::DataInputQueue::getName (C++ *function*), 215  
 dai::DataInputQueue::send (C++ *function*), 215  
 dai::DataInputQueue::setBlocking (C++ *function*), 214  
 dai::DataInputQueue::setMaxDataSize (C++ *function*), 214  
 dai::DataInputQueue::setMaxSize (C++ *function*), 214  
 dai::DataOutputQueue (C++ *class*), 215  
 dai::DataOutputQueue::addCallback (C++ *function*), 216  
 dai::DataOutputQueue::CallbackId (C++ *type*), 216



dai::DataOutputQueue::front (C++ function), 217  
 dai::DataOutputQueue::get (C++ function), 217  
 dai::DataOutputQueue::getAll (C++ function), 218  
 dai::DataOutputQueue::getBlocking (C++ function), 216  
 dai::DataOutputQueue::getMaxSize (C++ function), 216  
 dai::DataOutputQueue::getName (C++ function), 216  
 dai::DataOutputQueue::has (C++ function), 217  
 dai::DataOutputQueue::removeCallback (C++ function), 216  
 dai::DataOutputQueue::setBlocking (C++ function), 216  
 dai::DataOutputQueue::setMaxSize (C++ function), 216  
 dai::DataOutputQueue::tryGet (C++ function), 217  
 dai::DataOutputQueue::tryGetAll (C++ function), 217, 218  
 dai::DatatypeEnum (C++ enum), 207  
 dai::DatatypeEnum::Buffer (C++ enumerator), 207  
 dai::DatatypeEnum::CameraControl (C++ enumerator), 207  
 dai::DatatypeEnum::ImageManipConfig (C++ enumerator), 207  
 dai::DatatypeEnum::ImgDetections (C++ enumerator), 207  
 dai::DatatypeEnum::ImgFrame (C++ enumerator), 207  
 dai::DatatypeEnum::NNData (C++ enumerator), 207  
 dai::DatatypeEnum::SpatialImgDetections (C++ enumerator), 207  
 dai::DatatypeEnum::SpatialLocationCalculatorCorrection (C++ enumerator), 208  
 dai::DatatypeEnum::SpatialLocationCalculatorData (C++ enumerator), 208  
 dai::DatatypeEnum::SystemInformation (C++ enumerator), 207  
 dai::DetectionNetworkProperties (C++ struct), 218  
 dai::DetectionNetworkProperties::classes (C++ member), 218  
 dai::DetectionNetworkProperties::nnFamily (C++ member), 218  
 dai::Device (C++ class), 218  
 dai::Device::~~Device (C++ function), 219  
 dai::Device::addLogCallback (C++ function), 220  
 dai::Device::close (C++ function), 223  
 dai::Device::DEFAULT\_SEARCH\_TIME (C++ member), 224  
 dai::Device::DEFAULT\_SYSTEM\_INFORMATION\_LOGGING\_RATE (C++ member), 224  
 dai::Device::Device (C++ function), 219  
 dai::Device::EVENT\_QUEUE\_MAXIMUM\_SIZE (C++ member), 224  
 dai::Device::getAllAvailableDevices (C++ function), 223  
 dai::Device::getAnyAvailableDevice (C++ function), 223  
 dai::Device::getChipTemperature (C++ function), 222  
 dai::Device::getCmxMemoryUsage (C++ function), 222  
 dai::Device::getDdrMemoryUsage (C++ function), 222  
 dai::Device::getDeviceByMxId (C++ function), 223  
 dai::Device::getEmbeddedDeviceBinary (C++ function), 223  
 dai::Device::getFirstAvailableDevice (C++ function), 223  
 dai::Device::getInputQueue (C++ function), 221  
 dai::Device::getInputQueueNames (C++ function), 221  
 dai::Device::getLeonCssCpuUsage (C++ function), 223  
 dai::Device::getLeonCssHeapUsage (C++ function), 222  
 dai::Device::getLeonMssCpuUsage (C++ function), 223  
 dai::Device::getLeonMssHeapUsage (C++ function), 222  
 dai::Device::getLogLevel (C++ function), 220  
 dai::Device::getLogOutputLevel (C++ function), 220  
 dai::Device::getOutputQueue (C++ function), 220  
 dai::Device::getOutputQueueNames (C++ function), 221  
 dai::Device::getQueueEvent (C++ function), 222  
 dai::Device::getQueueEvents (C++ function), 221, 222  
 dai::Device::getSystemInformationLoggingRate (C++ function), 220  
 dai::Device::isClosed (C++ function), 223  
 dai::Device::isPipelineRunning (C++ function), 219  
 dai::Device::removeLogCallback (C++ function), 220

tion), 220  
 dai::Device::setLogLevel (C++ function), 220  
 dai::Device::setLogOutputLevel (C++ function), 220  
 dai::Device::setSystemInformationLoggingRate (C++ function), 220  
 dai::Device::startPipeline (C++ function), 219  
 dai::DeviceBootloader (C++ class), 224  
 dai::DeviceBootloader::close (C++ function), 225  
 dai::DeviceBootloader::createDepthaiApplication (C++ function), 225  
 dai::DeviceBootloader::DeviceBootloader (C++ function), 224  
 dai::DeviceBootloader::flash (C++ function), 224  
 dai::DeviceBootloader::flashBootloader (C++ function), 224  
 dai::DeviceBootloader::flashDepthaiApplication (C++ function), 224  
 dai::DeviceBootloader::getAllAvailableDevices (C++ function), 225  
 dai::DeviceBootloader::getEmbeddedBootloaderBin (C++ function), 225  
 dai::DeviceBootloader::getEmbeddedBootloaderVersion (C++ function), 225  
 dai::DeviceBootloader::getFirstAvailableDevice (C++ function), 225  
 dai::DeviceBootloader::getVersion (C++ function), 225  
 dai::DeviceBootloader::isClosed (C++ function), 225  
 dai::DeviceBootloader::isEmbeddedVersion (C++ function), 225  
 dai::DeviceBootloader::saveDepthaiApplicationParameters (C++ function), 225  
 dai::DeviceBootloader::Version (C++ struct), 225  
 dai::DeviceBootloader::Version::toString (C++ function), 226  
 dai::DeviceBootloader::Version::Version (C++ function), 226  
 dai::DeviceInfo (C++ struct), 226  
 dai::GlobalProperties (C++ struct), 226  
 dai::GlobalProperties::leonCssFrequencyHz (C++ member), 226  
 dai::GlobalProperties::leonMssFrequencyHz (C++ member), 226  
 dai::ImageManipConfig (C++ class), 226  
 dai::ImageManipConfig::getCropXMax (C++ function), 228  
 dai::ImageManipConfig::getCropXMin (C++ function), 228  
 dai::ImageManipConfig::getCropYMax (C++ function), 228  
 dai::ImageManipConfig::getCropYMin (C++ function), 228  
 dai::ImageManipConfig::getResizeHeight (C++ function), 228  
 dai::ImageManipConfig::getResizeWidth (C++ function), 228  
 dai::ImageManipConfig::ImageManipConfig (C++ function), 226  
 dai::ImageManipConfig::isResizeThumbnail (C++ function), 228  
 dai::ImageManipConfig::setCenterCrop (C++ function), 226  
 dai::ImageManipConfig::setCropRect (C++ function), 226  
 dai::ImageManipConfig::setCropRotatedRect (C++ function), 226  
 dai::ImageManipConfig:: setFrameType (C++ function), 227  
 dai::ImageManipConfig::setHorizontalFlip (C++ function), 227  
 dai::ImageManipConfig::setKeepAspectRatio (C++ function), 228  
 dai::ImageManipConfig::setResize (C++ function), 227  
 dai::ImageManipConfig::setResizeThumbnail (C++ function), 227  
 dai::ImageManipConfig::setReusePreviousImage (C++ function), 228  
 dai::ImageManipConfig::setRotationDegrees (C++ function), 227  
 dai::ImageManipConfig::setRotationRadians (C++ function), 227  
 dai::ImageManipConfig::setSkipCurrentImage (C++ function), 228  
 dai::ImageManipConfig::setWarpBorderFillColor (C++ function), 227  
 dai::ImageManipConfig::setWarpBorderReplicatePixels (C++ function), 227  
 dai::ImageManipConfig::setWarpTransformFourPoints (C++ function), 227  
 dai::ImageManipConfig::setWarpTransformMatrix3x3 (C++ function), 227  
 dai::ImageManipProperties (C++ struct), 228  
 dai::ImageManipProperties::initialConfig (C++ member), 228  
 dai::ImageManipProperties::inputConfigSync (C++ member), 228  
 dai::ImageManipProperties::numFramesPool (C++ member), 228  
 dai::ImageManipProperties::outputFrameSize (C++ member), 228  
 dai::ImgDetection (C++ struct), 228

dai::ImgDetections (C++ class), 228  
 dai::ImgDetections::detections (C++ member), 229  
 dai::ImgDetections::ImgDetections (C++ function), 229  
 dai::ImgFrame (C++ class), 229  
 dai::ImgFrame::getCategory (C++ function), 229  
 dai::ImgFrame::getCvFrame (C++ function), 230  
 dai::ImgFrame::getFrame (C++ function), 230  
 dai::ImgFrame::getHeight (C++ function), 229  
 dai::ImgFrame::getInstanceNum (C++ function), 229  
 dai::ImgFrame::getSequenceNum (C++ function), 229  
 dai::ImgFrame::getTimestamp (C++ function), 229  
 dai::ImgFrame::getType (C++ function), 229  
 dai::ImgFrame::getWidth (C++ function), 229  
 dai::ImgFrame::ImgFrame (C++ function), 229  
 dai::ImgFrame::setCategory (C++ function), 229  
 dai::ImgFrame::setFrame (C++ function), 230  
 dai::ImgFrame::setHeight (C++ function), 230  
 dai::ImgFrame::setInstanceNum (C++ function), 229  
 dai::ImgFrame::setSequenceNum (C++ function), 229  
 dai::ImgFrame::setTimestamp (C++ function), 229  
 dai::ImgFrame::setType (C++ function), 230  
 dai::ImgFrame::setWidth (C++ function), 230  
 dai::initialize (C++ function), 208  
 dai::isDatatypeSubclassOf (C++ function), 208  
 dai::LockingQueue (C++ class), 230  
 dai::LOG\_DEFAULT\_PATTERN (C++ member), 208  
 dai::LogLevel (C++ enum), 208  
 dai::LogLevel::CRITICAL (C++ enumerator), 208  
 dai::LogLevel::DEBUG (C++ enumerator), 208  
 dai::LogLevel::ERR (C++ enumerator), 208  
 dai::LogLevel::INFO (C++ enumerator), 208  
 dai::LogLevel::OFF (C++ enumerator), 208  
 dai::LogLevel::TRACE (C++ enumerator), 208  
 dai::LogLevel::WARN (C++ enumerator), 208  
 dai::LogMessage (C++ struct), 230  
 dai::MemoryInfo (C++ struct), 230  
 dai::MonoCameraProperties (C++ struct), 230  
 dai::MonoCameraProperties::boardSocket (C++ member), 231  
 dai::MonoCameraProperties::fps (C++ member), 231  
 dai::MonoCameraProperties::imageOrientation (C++ member), 231  
 dai::MonoCameraProperties::resolution (C++ member), 231  
 dai::MonoCameraProperties::SensorResolution (C++ enum), 231  
 dai::MonoCameraProperties::SensorResolution::THE\_4 (C++ enumerator), 231  
 dai::MonoCameraProperties::SensorResolution::THE\_7 (C++ enumerator), 231  
 dai::MonoCameraProperties::SensorResolution::THE\_8 (C++ enumerator), 231  
 dai::MyConsumerProperties (C++ struct), 231  
 dai::MyConsumerProperties::processorPlacement (C++ member), 231  
 dai::MyProducerProperties (C++ struct), 231  
 dai::MyProducerProperties::message (C++ member), 231  
 dai::MyProducerProperties::processorPlacement (C++ member), 231  
 dai::NeuralNetworkProperties (C++ struct), 231  
 dai::NeuralNetworkProperties::blobSize (C++ member), 232  
 dai::NeuralNetworkProperties::blobUri (C++ member), 232  
 dai::NeuralNetworkProperties::numFrames (C++ member), 232  
 dai::NeuralNetworkProperties::numNCEPerThread (C++ member), 232  
 dai::NeuralNetworkProperties::numThreads (C++ member), 232  
 dai::NNData (C++ class), 232  
 dai::NNData::getAllLayerNames (C++ function), 232  
 dai::NNData::getAllLayers (C++ function), 232  
 dai::NNData::getFirstLayerFp16 (C++ function), 233  
 dai::NNData::getFirstLayerInt32 (C++ function), 233  
 dai::NNData::getFirstLayerUInt8 (C++ function), 233  
 dai::NNData::getLayer (C++ function), 232  
 dai::NNData::getLayerDatatype (C++ function), 233  
 dai::NNData::getLayerFp16 (C++ function), 233  
 dai::NNData::getLayerInt32 (C++ function), 233  
 dai::NNData::getLayerUInt8 (C++ function), 233  
 dai::NNData::hasLayer (C++ function), 233  
 dai::NNData::NNData (C++ function), 232

```

dai::NNData::setLayer (C++ function), 232
dai::Node (C++ class), 233
dai::node (C++ type), 245
dai::node::ColorCamera (C++ class), 245
dai::node::ColorCamera::ColorCamera
    (C++ function), 246
dai::node::ColorCamera::getBoardSocket
    (C++ function), 246
dai::node::ColorCamera::getCamId (C++
    function), 246
dai::node::ColorCamera::getColorOrder
    (C++ function), 246
dai::node::ColorCamera::getFp16 (C++
    function), 246
dai::node::ColorCamera::getFps (C++ func-
    tion), 246
dai::node::ColorCamera::getImageOrientation
    (C++ function), 246
dai::node::ColorCamera::getInterleaved
    (C++ function), 246
dai::node::ColorCamera::getPreviewHeight
    (C++ function), 247
dai::node::ColorCamera::getPreviewKeepAspect
    Ratio (C++ function), 248
dai::node::ColorCamera::getPreviewSize
    (C++ function), 247
dai::node::ColorCamera::getPreviewWidth
    (C++ function), 247
dai::node::ColorCamera::getResolution
    (C++ function), 246
dai::node::ColorCamera::getResolutionHeight
    (C++ function), 247
dai::node::ColorCamera::getResolutionSize
    (C++ function), 247
dai::node::ColorCamera::getResolutionWidth
    (C++ function), 247
dai::node::ColorCamera::getSensorCrop
    (C++ function), 247
dai::node::ColorCamera::getSensorCropX
    (C++ function), 247
dai::node::ColorCamera::getSensorCropY
    (C++ function), 247
dai::node::ColorCamera::getStillHeight
    (C++ function), 247
dai::node::ColorCamera::getStillSize
    (C++ function), 247
dai::node::ColorCamera::getStillWidth
    (C++ function), 247
dai::node::ColorCamera::getVideoHeight
    (C++ function), 247
dai::node::ColorCamera::getVideoSize
    (C++ function), 247
dai::node::ColorCamera::getVideoWidth
    (C++ function), 247
dai::node::ColorCamera::getWaitForConfigInput
    (C++ function), 247
dai::node::ColorCamera::initialControl
    (C++ member), 248
dai::node::ColorCamera::inputConfig
    (C++ member), 248
dai::node::ColorCamera::inputControl
    (C++ member), 248
dai::node::ColorCamera::preview (C++
    member), 248
dai::node::ColorCamera::sensorCenterCrop
    (C++ function), 247
dai::node::ColorCamera::setBoardSocket
    (C++ function), 246
dai::node::ColorCamera::setCamId (C++
    function), 246
dai::node::ColorCamera::setColorOrder
    (C++ function), 246
dai::node::ColorCamera::setFp16 (C++
    function), 246
dai::node::ColorCamera::setFps (C++ func-
    tion), 246
dai::node::ColorCamera::setImageOrientation
    (C++ function), 246
dai::node::ColorCamera::setInterleaved
    (C++ function), 246
dai::node::ColorCamera::setPreviewKeepAspectRat-
    io (C++ function), 248
dai::node::ColorCamera::setPreviewSize
    (C++ function), 246
dai::node::ColorCamera::setResolution
    (C++ function), 246
dai::node::ColorCamera::setSensorCrop
    (C++ function), 247
dai::node::ColorCamera::setStillSize
    (C++ function), 246
dai::node::ColorCamera::setVideoSize
    (C++ function), 246
dai::node::ColorCamera::setWaitForConfigInput
    (C++ function), 247
dai::node::ColorCamera::still (C++ mem-
    ber), 248
dai::node::ColorCamera::video (C++ mem-
    ber), 248
dai::Node::Connection (C++ struct), 234
dai::node::DetectionNetwork (C++ class),
    248
dai::node::DetectionNetwork::input (C++
    member), 249
dai::node::DetectionNetwork::out (C++
    member), 249
dai::node::DetectionNetwork::passthrough
    (C++ member), 249
dai::node::DetectionNetwork::setConfidenceThreshol-
    d

```

(C++ function), 249

dai::Node::getAssets (C++ function), 234

dai::Node::getInputs (C++ function), 234

dai::Node::getName (C++ function), 234

dai::Node::getOutputs (C++ function), 234

dai::Node::id (C++ member), 234

dai::Node::Id (C++ type), 234

dai::node::ImageManip (C++ class), 249

dai::node::ImageManip::initialConfig (C++ member), 249

dai::node::ImageManip::inputConfig (C++ member), 249

dai::node::ImageManip::inputImage (C++ member), 249

dai::node::ImageManip::out (C++ member), 249

dai::node::ImageManip::setMaxOutputFrameSize (C++ function), 249

dai::node::ImageManip::setNumFramesPool (C++ function), 249

dai::node::ImageManip::setWaitForConfigInput (C++ function), 249

dai::node::MobileNetDetectionNetwork (C++ class), 250

dai::node::MobileNetSpatialDetectionNetwork (C++ class), 250

dai::node::MonoCamera (C++ class), 250

dai::node::MonoCamera::getBoardSocket (C++ function), 250

dai::node::MonoCamera::getFps (C++ function), 250

dai::node::MonoCamera::getImageOrientation (C++ function), 250

dai::node::MonoCamera::getResolution (C++ function), 250

dai::node::MonoCamera::getResolutionHeight (C++ function), 250

dai::node::MonoCamera::getResolutionSized (C++ function), 250

dai::node::MonoCamera::getResolutionWidth (C++ function), 250

dai::node::MonoCamera::initialControl (C++ member), 251

dai::node::MonoCamera::inputControl (C++ member), 251

dai::node::MonoCamera::out (C++ member), 251

dai::node::MonoCamera::setBoardSocket (C++ function), 250

dai::node::MonoCamera::setFps (C++ function), 250

dai::node::MonoCamera::setImageOrientation (C++ function), 250

dai::node::MonoCamera::setResolution (C++ function), 250

(C++ function), 250

dai::node::MyProducer (C++ class), 251

dai::node::NeuralNetwork (C++ class), 251

dai::node::NeuralNetwork::getNumInferenceThreads (C++ function), 251

dai::node::NeuralNetwork::input (C++ member), 252

dai::node::NeuralNetwork::out (C++ member), 252

dai::node::NeuralNetwork::passthrough (C++ member), 252

dai::node::NeuralNetwork::setBlobPath (C++ function), 251

dai::node::NeuralNetwork::setNumInferenceThreads (C++ function), 251

dai::node::NeuralNetwork::setNumNCEPerInferenceThread (C++ function), 251

dai::node::NeuralNetwork::setNumPoolFrames (C++ function), 251

dai::node::SpatialDetectionNetwork (C++ class), 252

dai::node::SpatialDetectionNetwork::boundingBoxMapping (C++ member), 252

dai::node::SpatialDetectionNetwork::input (C++ member), 252

dai::node::SpatialDetectionNetwork::inputDepth (C++ member), 252

dai::node::SpatialDetectionNetwork::out (C++ member), 252

dai::node::SpatialDetectionNetwork::passthrough (C++ member), 252

dai::node::SpatialDetectionNetwork::passthroughDepth (C++ member), 252

dai::node::SpatialDetectionNetwork::setBoundingBox (C++ function), 252

dai::node::SpatialDetectionNetwork::setDepthLowerThreshold (C++ function), 252

dai::node::SpatialDetectionNetwork::setDepthUpperThreshold (C++ function), 252

dai::node::SpatialLocationCalculator (C++ class), 253

dai::node::SpatialLocationCalculator::initialConfig (C++ member), 253

dai::node::SpatialLocationCalculator::inputConfig (C++ member), 253

dai::node::SpatialLocationCalculator::inputDepth (C++ member), 253

dai::node::SpatialLocationCalculator::out (C++ member), 253

dai::node::SpatialLocationCalculator::passthroughDepth (C++ member), 253

dai::node::SpatialLocationCalculator::setWaitForConfigInput (C++ function), 253

dai::node::SPIOut (C++ class), 253



dai::node::SPIOut::input (C++ member), 254  
 dai::node::SPIOut::setBusId (C++ function), 253  
 dai::node::SPIOut::setStreamName (C++ function), 253  
 dai::node::StereoDepth (C++ class), 254  
 dai::node::StereoDepth::depth (C++ member), 255  
 dai::node::StereoDepth::disparity (C++ member), 255  
 dai::node::StereoDepth::left (C++ member), 255  
 dai::node::StereoDepth::loadCalibrationData (C++ function), 254  
 dai::node::StereoDepth::loadCalibrationFile (C++ function), 254  
 dai::node::StereoDepth::rectifiedLeft (C++ member), 255  
 dai::node::StereoDepth::rectifiedRight (C++ member), 255  
 dai::node::StereoDepth::right (C++ member), 255  
 dai::node::StereoDepth::setConfidenceThreshold (C++ function), 254  
 dai::node::StereoDepth::setEmptyCalibrationData (C++ function), 254  
 dai::node::StereoDepth::setExtendedDisparity (C++ function), 254  
 dai::node::StereoDepth::setInputResolution (C++ function), 254  
 dai::node::StereoDepth::setLeftRightCheck (C++ function), 254  
 dai::node::StereoDepth::setMedianFilter (C++ function), 254  
 dai::node::StereoDepth::setOutputDepth (C++ function), 255  
 dai::node::StereoDepth::setOutputRectified (C++ function), 255  
 dai::node::StereoDepth::setRectifyEdgeField (C++ function), 254  
 dai::node::StereoDepth::setRectifyMirrorFrame (C++ function), 254  
 dai::node::StereoDepth::setSubpixel (C++ function), 254  
 dai::node::StereoDepth::syncedLeft (C++ member), 255  
 dai::node::StereoDepth::syncedRight (C++ member), 255  
 dai::node::SystemLogger (C++ class), 255  
 dai::node::SystemLogger::out (C++ member), 256  
 dai::node::SystemLogger::setRate (C++ function), 255  
 dai::node::VideoEncoder (C++ class), 256  
 dai::node::VideoEncoder::bitstream (C++ member), 257  
 dai::node::VideoEncoder::getBitrate (C++ function), 257  
 dai::node::VideoEncoder::getBitrateKbps (C++ function), 257  
 dai::node::VideoEncoder::getFrameRate (C++ function), 257  
 dai::node::VideoEncoder::getHeight (C++ function), 257  
 dai::node::VideoEncoder::getKeyframeFrequency (C++ function), 257  
 dai::node::VideoEncoder::getNumBFrames (C++ function), 257  
 dai::node::VideoEncoder::getNumFramesPool (C++ function), 256  
 dai::node::VideoEncoder::getProfile (C++ function), 257  
 dai::node::VideoEncoder::getQuality (C++ function), 257  
 dai::node::VideoEncoder::getRateControlMode (C++ function), 257  
 dai::node::VideoEncoder::getSize (C++ function), 257  
 dai::node::VideoEncoder::getWidth (C++ function), 257  
 dai::node::VideoEncoder::input (C++ member), 257  
 dai::node::VideoEncoder::setBitrate (C++ function), 256  
 dai::node::VideoEncoder::setBitrateKbps (C++ function), 256  
 dai::node::VideoEncoder::setDefaultProfilePreset (C++ function), 256  
 dai::node::VideoEncoder::setFrameRate (C++ function), 257  
 dai::node::VideoEncoder::setKeyframeFrequency (C++ function), 256  
 dai::node::VideoEncoder::setNumBFrames (C++ function), 257  
 dai::node::VideoEncoder::setNumFramesPool (C++ function), 256  
 dai::node::VideoEncoder::setProfile (C++ function), 256  
 dai::node::VideoEncoder::setQuality (C++ function), 257  
 dai::node::VideoEncoder::setRateControlMode (C++ function), 256  
 dai::node::XLinkIn (C++ class), 257  
 dai::node::XLinkIn::getMaxDataSize (C++ function), 258  
 dai::node::XLinkIn::getNumFrames (C++ function), 258  
 dai::node::XLinkIn::getStreamName (C++

*function*), 258  
 dai::node::XLinkIn::out (C++ *member*), 258  
 dai::node::XLinkIn::setMaxDataSize (C++ *function*), 258  
 dai::node::XLinkIn::setNumFrames (C++ *function*), 258  
 dai::node::XLinkIn::setStreamName (C++ *function*), 258  
 dai::node::XLinkOut (C++ *class*), 258  
 dai::node::XLinkOut::getFpsLimit (C++ *function*), 258  
 dai::node::XLinkOut::getMetadataOnly (C++ *function*), 258  
 dai::node::XLinkOut::getStreamName (C++ *function*), 258  
 dai::node::XLinkOut::input (C++ *member*), 259  
 dai::node::XLinkOut::setFpsLimit (C++ *function*), 258  
 dai::node::XLinkOut::setMetadataOnly (C++ *function*), 258  
 dai::node::XLinkOut::setStreamName (C++ *function*), 258  
 dai::node::YoloDetectionNetwork (C++ *class*), 259  
 dai::node::YoloDetectionNetwork::setAnchorMasks (C++ *function*), 259  
 dai::node::YoloDetectionNetwork::setAnchorSize (C++ *function*), 259  
 dai::node::YoloDetectionNetwork::setCoordinateSize (C++ *function*), 259  
 dai::node::YoloDetectionNetwork::setIoThreshold (C++ *function*), 259  
 dai::node::YoloDetectionNetwork::setNumClasses (C++ *function*), 259  
 dai::node::YoloSpatialDetectionNetwork (C++ *class*), 259  
 dai::node::YoloSpatialDetectionNetwork::defineAnchorMasks (C++ *function*), 259  
 dai::node::YoloSpatialDetectionNetwork::defineAnchorSize (C++ *function*), 259  
 dai::node::YoloSpatialDetectionNetwork::setCoordinateSize (C++ *function*), 259  
 dai::node::YoloSpatialDetectionNetwork::setIoThreshold (C++ *function*), 259  
 dai::node::YoloSpatialDetectionNetwork::setNumClasses (C++ *function*), 259  
 dai::NodeConnectionSchema (C++ *struct*), 234  
 dai::NodeIoInfo (C++ *struct*), 234  
 dai::NodeObjInfo (C++ *struct*), 234  
 dai::OpenVINO (C++ *class*), 234  
 dai::OpenVINO::areVersionsBlobCompatible (C++ *function*), 235  
 dai::OpenVINO::getBlobLatestSupportedVersion (C++ *function*), 235  
 dai::OpenVINO::getBlobSupportedVersions (C++ *function*), 235  
 dai::OpenVINO::getVersionName (C++ *function*), 235  
 dai::OpenVINO::getVersions (C++ *function*), 235  
 dai::OpenVINO::parseVersionName (C++ *function*), 235  
 dai::OpenVINO::Version (C++ *enum*), 234  
 dai::OpenVINO::Version::VERSION\_2020\_1 (C++ *enumerator*), 234  
 dai::OpenVINO::Version::VERSION\_2020\_2 (C++ *enumerator*), 234  
 dai::OpenVINO::Version::VERSION\_2020\_3 (C++ *enumerator*), 234  
 dai::OpenVINO::Version::VERSION\_2020\_4 (C++ *enumerator*), 234  
 dai::OpenVINO::Version::VERSION\_2021\_1 (C++ *enumerator*), 234  
 dai::OpenVINO::Version::VERSION\_2021\_2 (C++ *enumerator*), 234  
 dai::Pimpl (C++ *class*), 235  
 dai::Pipeline (C++ *class*), 235  
 dai::Pipeline::create (C++ *function*), 235  
 dai::Pipeline::DEFAULT\_OPENVINO\_VERSION (C++ *member*), 237  
 dai::Pipeline::getAllAssets (C++ *function*), 236  
 dai::Pipeline::getAllNodes (C++ *function*), 236  
 dai::Pipeline::getAssetManager (C++ *function*), 236  
 dai::Pipeline::getConnectionMap (C++ *function*), 236  
 dai::Pipeline::getConnections (C++ *function*), 236  
 dai::Pipeline::getGlobalProperties (C++ *function*), 235  
 dai::Pipeline::getNode (C++ *function*), 236  
 dai::Pipeline::getNodeMap (C++ *function*), 236  
 dai::Pipeline::getPipelineSchema (C++ *function*), 235  
 dai::Pipeline::link (C++ *function*), 236  
 dai::Pipeline::Pipeline (C++ *function*), 235  
 dai::Pipeline::remove (C++ *function*), 235  
 dai::Pipeline::setOpenVINOVersion (C++ *function*), 236  
 dai::Pipeline::unlink (C++ *function*), 236  
 dai::PipelineImpl (C++ *class*), 237  
 dai::PipelineSchema (C++ *struct*), 237  
 dai::Point2f (C++ *struct*), 237  
 dai::Point3f (C++ *struct*), 237

dai::ProcessorType (C++ *enum*), 207  
 dai::ProcessorType::LOS (C++ *enumerator*), 207  
 dai::ProcessorType::LRT (C++ *enumerator*), 207  
 dai::RawBuffer (C++ *struct*), 237  
 dai::RawCameraControl (C++ *struct*), 237  
 dai::RawCameraControl::lensPosition (C++ *member*), 237  
 dai::RawCameraControl::ManualExposureParams (C++ *struct*), 237  
 dai::RawCameraControl::RegionParams (C++ *struct*), 237  
 dai::RawImageManipConfig (C++ *struct*), 237  
 dai::RawImageManipConfig::CropConfig (C++ *struct*), 237  
 dai::RawImageManipConfig::CropRect (C++ *struct*), 237  
 dai::RawImageManipConfig::FormatConfig (C++ *struct*), 237  
 dai::RawImageManipConfig::ResizeConfig (C++ *struct*), 237  
 dai::RawImageManipConfig::ResizeConfig::keepAspectRatio (C++ *member*), 238  
 dai::RawImgDetections (C++ *struct*), 238  
 dai::RawImgFrame (C++ *struct*), 238  
 dai::RawImgFrame::Specs (C++ *struct*), 238  
 dai::RawNNData (C++ *struct*), 238  
 dai::RawSpatialImgDetections (C++ *struct*), 238  
 dai::RawSpatialLocationCalculatorConfig (C++ *struct*), 238  
 dai::RawSpatialLocations (C++ *struct*), 238  
 dai::RawSystemInformation (C++ *struct*), 238  
 dai::RawSystemInformation::chipTemperature (C++ *member*), 238  
 dai::RawSystemInformation::cmxMemoryUsage (C++ *member*), 238  
 dai::RawSystemInformation::ddrMemoryUsage (C++ *member*), 238  
 dai::RawSystemInformation::leonCssCpuUsage (C++ *member*), 238  
 dai::RawSystemInformation::leonCssMemoryUsage (C++ *member*), 238  
 dai::RawSystemInformation::leonMssCpuUsage (C++ *member*), 238  
 dai::RawSystemInformation::leonMssMemoryUsage (C++ *member*), 238  
 dai::Rect (C++ *struct*), 238  
 dai::Rect::area (C++ *function*), 239  
 dai::Rect::bottomRight (C++ *function*), 239  
 dai::Rect::contains (C++ *function*), 239  
 dai::Rect::denormalize (C++ *function*), 239  
 dai::Rect::empty (C++ *function*), 239  
 dai::Rect::isNormalized (C++ *function*), 239  
 dai::Rect::normalize (C++ *function*), 239  
 dai::Rect::size (C++ *function*), 239  
 dai::Rect::topLeft (C++ *function*), 239  
 dai::RotatedRect (C++ *struct*), 239  
 dai::RotatedRect::angle (C++ *member*), 239  
 dai::Size2f (C++ *struct*), 239  
 dai::SpatialDetectionNetworkProperties (C++ *struct*), 239  
 dai::SpatialImgDetection (C++ *struct*), 239  
 dai::SpatialImgDetections (C++ *class*), 239  
 dai::SpatialImgDetections::detections (C++ *member*), 240  
 dai::SpatialImgDetections::SpatialImgDetections (C++ *function*), 240  
 dai::SpatialLocationCalculatorConfig (C++ *class*), 240  
 dai::SpatialLocationCalculatorConfig::addROI (C++ *function*), 240  
 dai::SpatialLocationCalculatorConfig::getConfigData (C++ *function*), 240  
 dai::SpatialLocationCalculatorConfig::setROIs (C++ *function*), 240  
 dai::SpatialLocationCalculatorConfig::keepAspectRatio (C++ *function*), 240  
 dai::SpatialLocationCalculatorConfig::SpatialLocationCalculatorConfig (C++ *function*), 240  
 dai::SpatialLocationCalculatorConfigData (C++ *struct*), 240  
 dai::SpatialLocationCalculatorConfigThresholds (C++ *struct*), 240  
 dai::SpatialLocationCalculatorData (C++ *class*), 240  
 dai::SpatialLocationCalculatorData::getSpatialLocationCalculatorData (C++ *function*), 240  
 dai::SpatialLocationCalculatorData::SpatialLocationCalculatorData (C++ *function*), 240  
 dai::SpatialLocationCalculatorProperties (C++ *struct*), 240  
 dai::SpatialLocationCalculatorProperties::inputConfidence (C++ *member*), 241  
 dai::SpatialLocations (C++ *struct*), 241  
 dai::SPIOutProperties (C++ *struct*), 241  
 dai::SPIOutProperties::busId (C++ *member*), 241  
 dai::SPIOutProperties::streamName (C++ *member*), 241  
 dai::StereoDepthProperties (C++ *struct*), 241  
 dai::StereoDepthProperties::calibration (C++ *member*), 241  
 dai::StereoDepthProperties::confidenceThreshold (C++ *member*), 241  
 dai::StereoDepthProperties::enableExtendedDisparity (C++ *member*), 241  
 dai::StereoDepthProperties::enableLeftRightCheck (C++ *member*), 241



dai::StereoDepthProperties::enableOutputDepth (C++ member), 243  
 (C++ member), 242  
 dai::StereoDepthProperties::enableOutputRectified (C++ enumerator), 243  
 (C++ member), 242  
 dai::StereoDepthProperties::enableSubpixel (C++ enumerator), 243  
 (C++ member), 241  
 dai::StereoDepthProperties::height (C++ member), 242  
 dai::StereoDepthProperties::median (C++ member), 241  
 dai::StereoDepthProperties::MedianFilter (C++ enum), 241  
 dai::StereoDepthProperties::MedianFilter::KERNEL\_3x3 (C++ member), 243  
 (C++ enumerator), 241  
 dai::StereoDepthProperties::MedianFilter::KERNEL\_5x5 (C++ enum), 242  
 (C++ enumerator), 241  
 dai::StereoDepthProperties::MedianFilter::KERNEL\_7x7 (C++ enumerator), 242  
 (C++ enumerator), 241  
 dai::StereoDepthProperties::MedianFilter::MEDIAN\_OFF (C++ enumerator), 242  
 (C++ enumerator), 241  
 dai::StereoDepthProperties::rectifyEdgeFillColor (C++ member), 243  
 (C++ member), 242  
 dai::StereoDepthProperties::rectifyMirrorFrame (C++ member), 242  
 (C++ member), 242  
 dai::StereoDepthProperties::width (C++ member), 242  
 (C++ member), 242  
 dai::SystemInformation (C++ class), 242  
 dai::SystemInformation::SystemInformation (C++ function), 242  
 dai::SystemLoggerProperties (C++ struct), 242  
 dai::SystemLoggerProperties::rateHz (C++ member), 242  
 dai::TensorInfo (C++ struct), 242  
 dai::Timestamp (C++ struct), 242  
 dai::VideoEncoderProperties (C++ struct), 242  
 dai::VideoEncoderProperties::bitrate (C++ member), 243  
 dai::VideoEncoderProperties::frameRate (C++ member), 243  
 dai::VideoEncoderProperties::height (C++ member), 243  
 dai::VideoEncoderProperties::keyframeFrequency (C++ member), 243  
 (C++ member), 243  
 dai::VideoEncoderProperties::maxBitrate (C++ member), 243  
 dai::VideoEncoderProperties::numBFrames (C++ member), 243  
 dai::VideoEncoderProperties::numFramesPerKeyframe (C++ member), 243  
 (C++ member), 243  
 dai::VideoEncoderProperties::Profile (C++ enum), 242  
 dai::VideoEncoderProperties::profile (C++ member), 242  
 dai::VideoEncoderProperties::Profile::H264\_BASELINE (C++ enumerator), 243  
 dai::VideoEncoderProperties::Profile::H264\_HIGH (C++ enumerator), 243  
 dai::VideoEncoderProperties::Profile::H264\_MAIN (C++ enumerator), 243  
 dai::VideoEncoderProperties::Profile::H265\_MAIN (C++ enumerator), 243  
 dai::VideoEncoderProperties::Profile::MJPEG (C++ enumerator), 243  
 dai::VideoEncoderProperties::quality (C++ member), 243  
 dai::VideoEncoderProperties::RateControlMode (C++ member), 243  
 dai::VideoEncoderProperties::RateControlMode::CBR (C++ member), 243  
 dai::VideoEncoderProperties::RateControlMode::VBR (C++ member), 243  
 dai::VideoEncoderProperties::rateCtrlMode (C++ member), 243  
 dai::VideoEncoderProperties::width (C++ member), 243  
 dai::XLINK\_CHANNEL\_LOG (C++ member), 208  
 dai::XLINK\_CHANNEL\_MAIN\_RPC (C++ member), 208  
 dai::XLINK\_CHANNEL\_PIPELINE\_CONFIG (C++ member), 208  
 dai::XLINK\_CHANNEL\_TIMESYNC (C++ member), 208  
 dai::XLINK\_USB\_BUFFER\_MAX\_SIZE (C++ member), 208  
 dai::XLINK\_WATCHDOG\_TIMEOUT (C++ member), 208  
 dai::XLinkConnection (C++ class), 243  
 dai::XLinkConnection::close (C++ function), 244  
 dai::XLinkConnection::isClosed (C++ function), 244  
 dai::XLinkInProperties (C++ struct), 244  
 dai::XLinkInProperties::maxDataSize (C++ member), 244  
 dai::XLinkInProperties::numFrames (C++ member), 244  
 dai::XLinkInProperties::streamName (C++ member), 244  
 dai::XLinkOutProperties (C++ struct), 244  
 dai::XLinkOutProperties::maxFpsLimit (C++ member), 244  
 dai::XLinkOutProperties::metadataOnly (C++ member), 244  
 dai::XLinkOutProperties::streamName (C++ member), 244  
 dai::XLinkStream (C++ class), 244

- [data \(\) \(depthai.Asset property\), 101](#)  
[data \(\) \(depthai.RawBuffer property\), 164](#)  
[DataInputQueue \(class in depthai\), 121](#)  
[DataOutputQueue \(class in depthai\), 122](#)  
[dataType \(\) \(depthai.TensorInfo property\), 195](#)  
[DAYLIGHT \(depthai.CameraControl.AutoWhiteBalanceMode attribute\), 107](#)  
[DAYLIGHT \(depthai.RawCameraControl.AutoWhiteBalanceMode attribute\), 166](#)  
[ddrMemoryUsage \(\) \(depthai.RawSystemInformation property\), 178](#)  
[ddrMemoryUsage \(\) \(depthai.SystemInformation property\), 192](#)  
[DEBUG \(depthai.LogLevel attribute\), 147](#)  
[denormalize \(\) \(depthai.Rect method\), 179](#)  
[depth \(\) \(depthai.StereoDepth property\), 188](#)  
[depthai module, 99](#)  
[depthAverage \(\) \(depthai.SpatialLocations property\), 187](#)  
[depthThresholds \(\) \(depthai.SpatialDetectionNetworkProperties property\), 182](#)  
[depthThresholds \(\) \(depthai.SpatialLocationCalculatorConfigData property\), 185](#)  
[desc \(\) \(depthai.DeviceInfo property\), 135](#)  
[detectedBBScaleFactor \(\) \(depthai.SpatialDetectionNetworkProperties property\), 182](#)  
[DetectionNetwork \(class in depthai\), 124](#)  
[DetectionNetworkProperties \(class in depthai\), 125](#)  
[detections \(\) \(depthai.ImgDetections property\), 141](#)  
[detections \(\) \(depthai.RawImgDetections property\), 173](#)  
[detections \(\) \(depthai.RawSpatialImgDetections property\), 177](#)  
[detections \(\) \(depthai.SpatialImgDetections property\), 183](#)  
[Device \(class in depthai\), 125](#)  
[DeviceBootloader \(class in depthai\), 132](#)  
[DeviceBootloader.Version \(class in depthai\), 133](#)  
[DeviceDesc \(class in depthai\), 135](#)  
[DeviceInfo \(class in depthai\), 135](#)  
[dims \(\) \(depthai.TensorInfo property\), 195](#)  
[disparity \(\) \(depthai.StereoDepth property\), 188](#)  
[dss \(\) \(depthai.ChipTemperature property\), 113](#)
- E**
- [EDOF \(depthai.CameraControl.AutoFocusMode attribute\), 106](#)  
[EDOF \(depthai.RawCameraControl.AutoFocusMode attribute\), 166](#)  
[empty \(\) \(depthai.Rect method\), 179](#)  
[enableCenterCropRectangle \(\) \(depthai.RawImageManipConfig.CropConfig property\), 170](#)  
[enableCrop \(\) \(depthai.RawImageManipConfig property\), 172](#)  
[enableExtendedDisparity \(\) \(depthai.StereoDepthProperties property\), 191](#)  
[enableFormat \(\) \(depthai.RawImageManipConfig property\), 172](#)  
[enableLeftRightCheck \(\) \(depthai.StereoDepthProperties property\), 191](#)  
[enableOutputDepth \(\) \(depthai.StereoDepthProperties property\), 191](#)  
[enableOutputRectified \(\) \(depthai.StereoDepthProperties property\), 191](#)  
[enableResize \(\) \(depthai.RawImageManipConfig property\), 172](#)  
[enableRotatedRect \(\) \(depthai.RawImageManipConfig.CropConfig property\), 170](#)  
[enableRotation \(\) \(depthai.RawImageManipConfig.ResizeConfig property\), 172](#)  
[enableSubpixel \(\) \(depthai.StereoDepthProperties property\), 191](#)  
[enableWarp4pt \(\) \(depthai.RawImageManipConfig.ResizeConfig property\), 172](#)  
[enableWarpMatrix \(\) \(depthai.RawImageManipConfig.ResizeConfig property\), 172](#)  
[ERR \(depthai.LogLevel attribute\), 147](#)
- F**
- [FACE\\_PRIORITY \(depthai.CameraControl.SceneMode attribute\), 110](#)  
[FACE\\_PRIORITY \(depthai.RawCameraControl.SceneMode attribute\), 169](#)  
[fb \(\) \(depthai.RawImgFrame property\), 177](#)  
[FIREWORKS \(depthai.CameraControl.SceneMode attribute\), 110](#)  
[FIREWORKS \(depthai.RawCameraControl.SceneMode attribute\), 169](#)  
[flash \(\) \(depthai.DeviceBootloader method\), 134](#)  
[flashBootloader \(\) \(depthai.DeviceBootloader method\), 134](#)  
[flashDepthaiApplicationPackage \(\) \(depthai.DeviceBootloader method\), 134](#)

[flipHorizontal\(\)](#) (*depthai.RawImageManipConfig.FormatConfig* property), 171  
[FLUORESCENT](#) (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 107  
[FLUORESCENT](#) (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 166  
[formatConfig\(\)](#) (*depthai.RawImageManipConfig.FormatConfig* property), 172  
[FP16](#) (*depthai.TensorInfo.DataType* attribute), 194  
[FP32](#) (*depthai.TensorInfo.DataType* attribute), 194  
[fps\(\)](#) (*depthai.ColorCameraProperties* property), 120  
[fps\(\)](#) (*depthai.MonoCameraProperties* property), 151

## G

[get\(\)](#) (*depthai.AssetManager* method), 102  
[get\(\)](#) (*depthai.DataOutputQueue* method), 123  
[getAll\(\)](#) (*depthai.AssetManager* method), 102  
[getAll\(\)](#) (*depthai.DataOutputQueue* method), 123  
[getAllAssets\(\)](#) (*depthai.Pipeline* method), 162  
[getAllAvailableDevices\(\)](#) (*depthai.Device* static method), 128  
[getAllAvailableDevices\(\)](#) (*depthai.DeviceBootloader* static method), 134  
[getAllConnectedDevices\(\)](#) (*depthai.XLinkConnection* static method), 202  
[getAllLayerNames\(\)](#) (*depthai.NNData* method), 152  
[getAllLayers\(\)](#) (*depthai.NNData* method), 152  
[getAllNodes\(\)](#) (*depthai.Pipeline* method), 162  
[getAnyAvailableDevice\(\)](#) (*depthai.Device* static method), 128  
[getAssetManager\(\)](#) (*depthai.Pipeline* method), 162  
[getAssets\(\)](#) (*depthai.Node* method), 158  
[getBitrate\(\)](#) (*depthai.VideoEncoder* method), 197  
[getBitrateKbps\(\)](#) (*depthai.VideoEncoder* method), 197  
[getBlocking\(\)](#) (*depthai.DataInputQueue* method), 121  
[getBlocking\(\)](#) (*depthai.DataOutputQueue* method), 123  
[getBlocking\(\)](#) (*depthai.Node.Input* method), 157  
[getBoardSocket\(\)](#) (*depthai.ColorCamera* method), 116  
[getBoardSocket\(\)](#) (*depthai.MonoCamera* method), 149  
[getCamId\(\)](#) (*depthai.ColorCamera* method), 116  
[getCamId\(\)](#) (*depthai.MonoCamera* method), 149  
[getCaptureStill\(\)](#) (*depthai.CameraControl* method), 110  
[getCategory\(\)](#) (*depthai.ImgFrame* method), 145  
[getChipTemperature\(\)](#) (*depthai.Device* method), 128  
[getDdrMemoryUsage\(\)](#) (*depthai.Device* method), 128  
[getColorOrder\(\)](#) (*depthai.ColorCamera* method), 116  
[getGpuUsageData\(\)](#) (*depthai.SpatialLocationCalculatorConfig* method), 185  
[getConnectionMap\(\)](#) (*depthai.Pipeline* method), 162  
[getConnections\(\)](#) (*depthai.Node.Output* method), 157  
[getConnections\(\)](#) (*depthai.Pipeline* method), 162  
[getCropXMax\(\)](#) (*depthai.ImageManipConfig* method), 138  
[getCropXMin\(\)](#) (*depthai.ImageManipConfig* method), 138  
[getCropYMax\(\)](#) (*depthai.ImageManipConfig* method), 138  
[getCropYMin\(\)](#) (*depthai.ImageManipConfig* method), 138  
[getCvFrame\(\)](#) (*depthai.ImgFrame* method), 145  
[getData\(\)](#) (*depthai.Buffer* method), 103  
[getDdrMemoryUsage\(\)](#) (*depthai.Device* method), 128  
[getDeviceByMxId\(\)](#) (*depthai.Device* static method), 128  
[getDeviceByMxId\(\)](#) (*depthai.XLinkConnection* static method), 202  
[getEmbeddedBootloaderBinary\(\)](#) (*depthai.DeviceBootloader* static method), 134  
[getEmbeddedBootloaderVersion\(\)](#) (*depthai.DeviceBootloader* static method), 134  
[getEmbeddedDeviceBinary\(\)](#) (*depthai.Device* static method), 128  
[getFirstAvailableDevice\(\)](#) (*depthai.Device* static method), 129  
[getFirstAvailableDevice\(\)](#) (*depthai.DeviceBootloader* static method), 134  
[getFirstDevice\(\)](#) (*depthai.XLinkConnection* static method), 202  
[getFirstLayerFp16\(\)](#) (*depthai.NNData* method), 152  
[getFirstLayerInt32\(\)](#) (*depthai.NNData* method), 152  
[getFirstLayerUInt8\(\)](#) (*depthai.NNData* method), 152  
[getFp16\(\)](#) (*depthai.ColorCamera* method), 116  
[getFps\(\)](#) (*depthai.ColorCamera* method), 116  
[getFps\(\)](#) (*depthai.MonoCamera* method), 149  
[getFpsLimit\(\)](#) (*depthai.XLinkOut* method), 204  
[getFrame\(\)](#) (*depthai.ImgFrame* method), 145  
[getFrameRate\(\)](#) (*depthai.VideoEncoder* method),

197

`getGlobalProperties()` (*depthai.Pipeline method*), 162

`getHeight()` (*depthai.ImgFrame method*), 145

`getHeight()` (*depthai.VideoEncoder method*), 197

`getImageOrientation()` (*depthai.ColorCamera method*), 116

`getImageOrientation()` (*depthai.MonoCamera method*), 149

`getInputQueue()` (*depthai.Device method*), 129

`getInputQueueNames()` (*depthai.Device method*), 129

`getInputs()` (*depthai.Node method*), 158

`getInstanceNum()` (*depthai.ImgFrame method*), 146

`getInterleaved()` (*depthai.ColorCamera method*), 116

`getKeyframeFrequency()` (*depthai.VideoEncoder method*), 198

`getLayer()` (*depthai.NNData method*), 152

`getLayerDatatype()` (*depthai.NNData method*), 152

`getLayerFp16()` (*depthai.NNData method*), 152

`getLayerInt32()` (*depthai.NNData method*), 152

`getLayerUInt8()` (*depthai.NNData method*), 153

`getLeonCssCpuUsage()` (*depthai.Device method*), 129

`getLeonCssHeapUsage()` (*depthai.Device method*), 129

`getLeonMssCpuUsage()` (*depthai.Device method*), 129

`getLeonMssHeapUsage()` (*depthai.Device method*), 129

`getLogLevel()` (*depthai.Device method*), 129

`getLogOutputLevel()` (*depthai.Device method*), 130

`getMaxDataSize()` (*depthai.XLinkIn method*), 203

`getMaxSize()` (*depthai.DataInputQueue method*), 121

`getMaxSize()` (*depthai.DataOutputQueue method*), 123

`getMetadataOnly()` (*depthai.XLinkOut method*), 204

`getMxId()` (*depthai.DeviceInfo method*), 135

`getName()` (*depthai.DataInputQueue method*), 121

`getName()` (*depthai.DataOutputQueue method*), 123

`getName()` (*depthai.Node method*), 158

`getNode()` (*depthai.Pipeline method*), 162

`getNodeMap()` (*depthai.Pipeline method*), 162

`getNumBframes()` (*depthai.VideoEncoder method*), 198

`getNumFrames()` (*depthai.XLinkIn method*), 203

`getNumFramesPool()` (*depthai.VideoEncoder method*), 198

`getNumInferenceThreads()` (*depthai.NeuralNetwork method*), 154

`getOutputQueue()` (*depthai.Device method*), 130

`getOutputQueueNames()` (*depthai.Device method*), 130

`getOutputs()` (*depthai.Node method*), 158

`getPreviewHeight()` (*depthai.ColorCamera method*), 116

`getPreviewKeepAspectRatio()` (*depthai.ColorCamera method*), 116

`getPreviewSize()` (*depthai.ColorCamera method*), 116

`getPreviewWidth()` (*depthai.ColorCamera method*), 116

`getProfile()` (*depthai.VideoEncoder method*), 198

`getQuality()` (*depthai.VideoEncoder method*), 198

`getQueueEvent()` (*depthai.Device method*), 130

`getQueueEvents()` (*depthai.Device method*), 131

`getQueueSize()` (*depthai.Node.Input method*), 157

`getRateControlMode()` (*depthai.VideoEncoder method*), 198

`getRaw()` (*depthai.ADatatype method*), 101

`getResizeHeight()` (*depthai.ImageManipConfig method*), 139

`getResizeWidth()` (*depthai.ImageManipConfig method*), 139

`getResolution()` (*depthai.ColorCamera method*), 116

`getResolution()` (*depthai.MonoCamera method*), 149

`getResolutionHeight()` (*depthai.ColorCamera method*), 116

`getResolutionHeight()` (*depthai.MonoCamera method*), 149

`getResolutionSize()` (*depthai.ColorCamera method*), 116

`getResolutionSize()` (*depthai.MonoCamera method*), 149

`getResolutionWidth()` (*depthai.ColorCamera method*), 116

`getResolutionWidth()` (*depthai.MonoCamera method*), 149

`getSensorCrop()` (*depthai.ColorCamera method*), 116

`getSensorCropX()` (*depthai.ColorCamera method*), 117

`getSensorCropY()` (*depthai.ColorCamera method*), 117

`getSequenceNum()` (*depthai.ImgFrame method*), 146

`getSize()` (*depthai.VideoEncoder method*), 198

`getSpatialLocations()` (*depthai.SpatialLocationCalculatorData method*), 186

`getStillHeight()` (*depthai.ColorCamera method*), 117



[getStillSize\(\)](#) (*depthai.ColorCamera method*), 117  
[getStillWidth\(\)](#) (*depthai.ColorCamera method*), 117  
[getStreamName\(\)](#) (*depthai.XLinkIn method*), 203  
[getStreamName\(\)](#) (*depthai.XLinkOut method*), 204  
[getSystemInformationLoggingRate\(\)](#) (*depthai.Device method*), 131  
[getTimestamp\(\)](#) (*depthai.ImgFrame method*), 146  
[getType\(\)](#) (*depthai.ImgFrame method*), 146  
[getVersion\(\)](#) (*depthai.DeviceBootloader method*), 134  
[getVersionName\(\)](#) (*depthai.OpenVINO static method*), 160  
[getVersions\(\)](#) (*depthai.OpenVINO static method*), 160  
[getVideoHeight\(\)](#) (*depthai.ColorCamera method*), 117  
[getVideoSize\(\)](#) (*depthai.ColorCamera method*), 117  
[getVideoWidth\(\)](#) (*depthai.ColorCamera method*), 117  
[getWaitForConfigInput\(\)](#) (*depthai.ColorCamera method*), 117  
[getWidth\(\)](#) (*depthai.ImgFrame method*), 146  
[getWidth\(\)](#) (*depthai.VideoEncoder method*), 198  
[GlobalProperties](#) (*class in depthai*), 135  
[GRAY8](#) (*depthai.ImgFrame.Type attribute*), 144  
[GRAY8](#) (*depthai.RawImgFrame.Type attribute*), 176  
[GRAYF16](#) (*depthai.ImgFrame.Type attribute*), 144  
[GRAYF16](#) (*depthai.RawImgFrame.Type attribute*), 176

## H

[H](#) (*depthai.TensorInfo.StorageOrder attribute*), 195  
[H264\\_BASELINE](#) (*depthai.VideoEncoderProperties.Profile attribute*), 200  
[H264\\_HIGH](#) (*depthai.VideoEncoderProperties.Profile attribute*), 200  
[H264\\_MAIN](#) (*depthai.VideoEncoderProperties.Profile attribute*), 200  
[H265\\_MAIN](#) (*depthai.VideoEncoderProperties.Profile attribute*), 200  
[has\(\)](#) (*depthai.DataOutputQueue method*), 123  
[hasLayer\(\)](#) (*depthai.NNData method*), 153  
[HCW](#) (*depthai.TensorInfo.StorageOrder attribute*), 195  
[HDR](#) (*depthai.ImgFrame.Type attribute*), 145  
[HDR](#) (*depthai.RawImgFrame.Type attribute*), 176  
[height\(\)](#) (*depthai.ImgFrame.Specs property*), 142  
[height\(\)](#) (*depthai.RawImageManipConfig.ResizeConfig property*), 172  
[height\(\)](#) (*depthai.RawImgFrame.Specs property*), 173  
[height\(\)](#) (*depthai.Rect property*), 179  
[height\(\)](#) (*depthai.Size2f property*), 181  
[height\(\)](#) (*depthai.StereoDepthProperties property*), 191  
[height\(\)](#) (*depthai.VideoEncoderProperties property*), 201  
[HORIZONTAL\\_MIRROR](#) (*depthai.CameraImageOrientation attribute*), 113  
[HWC](#) (*depthai.TensorInfo.StorageOrder attribute*), 195

## I

[I8](#) (*depthai.TensorInfo.DataType attribute*), 194  
[id\(\)](#) (*depthai.Node property*), 158  
[ImageManip](#) (*class in depthai*), 136  
[ImageManipConfig](#) (*class in depthai*), 137  
[ImgDetection](#) (*class in depthai*), 140  
[ImgDetections](#) (*class in depthai*), 141  
[ImgFrame](#) (*class in depthai*), 141  
[ImgFrame.Specs](#) (*class in depthai*), 142  
[ImgFrame.Type](#) (*class in depthai*), 143  
[INCANDESCENT](#) (*depthai.CameraControl.AutoWhiteBalanceMode attribute*), 107  
[INCANDESCENT](#) (*depthai.RawCameraControl.AutoWhiteBalanceMode attribute*), 166  
[INFO](#) (*depthai.LogLevel attribute*), 147  
[initialConfig\(\)](#) (*depthai.ImageManip property*), 136  
[initialConfig\(\)](#) (*depthai.SpatialLocationCalculator property*), 184  
[initialControl\(\)](#) (*depthai.ColorCamera property*), 117  
[initialControl\(\)](#) (*depthai.ColorCameraProperties property*), 120  
[initialControl\(\)](#) (*depthai.MonoCamera property*), 149  
[initialControl\(\)](#) (*depthai.MonoCameraProperties property*), 151  
[input\(\)](#) (*depthai.DetectionNetwork property*), 125  
[input\(\)](#) (*depthai.NeuralNetwork property*), 154  
[input\(\)](#) (*depthai.SpatialDetectionNetwork property*), 182  
[input\(\)](#) (*depthai.SPIOut property*), 180  
[input\(\)](#) (*depthai.VideoEncoder property*), 198  
[input\(\)](#) (*depthai.XLinkOut property*), 204  
[inputConfig\(\)](#) (*depthai.ColorCamera property*), 117  
[inputConfig\(\)](#) (*depthai.ImageManip property*), 136  
[inputConfig\(\)](#) (*depthai.SpatialLocationCalculator property*), 184  
[inputConfigSync\(\)](#) (*depthai.SpatialLocationCalculatorProperties property*), 186  
[inputControl\(\)](#) (*depthai.ColorCamera property*), 117

[inputControl\(\)](#) (*depthai.MonoCamera* property), [149](#)  
[inputDepth\(\)](#) (*depthai.SpatialDetectionNetwork* property), [182](#)  
[inputDepth\(\)](#) (*depthai.SpatialLocationCalculator* property), [184](#)  
[inputId\(\)](#) (*depthai.Node.Connection* property), [156](#)  
[inputImage\(\)](#) (*depthai.ImageManip* property), [137](#)  
[inputName\(\)](#) (*depthai.Node.Connection* property), [156](#)  
[instanceNum\(\)](#) (*depthai.RawImgFrame* property), [177](#)  
[INT](#) (*depthai.TensorInfo.DataType* attribute), [194](#)  
[interleaved\(\)](#) (*depthai.ColorCameraProperties* property), [120](#)  
[iouThreshold\(\)](#) (*depthai.DetectionNetworkProperties* property), [125](#)  
[isEmbeddedVersion\(\)](#) (*depthai.DeviceBootloader* method), [134](#)  
[isNormalized\(\)](#) (*depthai.Rect* method), [179](#)  
[isPipelineRunning\(\)](#) (*depthai.Device* method), [132](#)  
[isResizeThumbnail\(\)](#) (*depthai.ImageManipConfig* method), [139](#)

## K

[keepAspectRatio\(\)](#) (*depthai.RawImageManipConfig.ResizeConfig* property), [172](#)  
[KERNEL\\_3x3](#) (*depthai.StereoDepthProperties.MedianFilter* attribute), [191](#)  
[KERNEL\\_5x5](#) (*depthai.StereoDepthProperties.MedianFilter* attribute), [191](#)  
[KERNEL\\_7x7](#) (*depthai.StereoDepthProperties.MedianFilter* attribute), [191](#)  
[key\(\)](#) (*depthai.Asset* property), [101](#)  
[keyframeFrequency\(\)](#) (*depthai.VideoEncoderProperties* property), [201](#)

[leonCssCpuUsage\(\)](#) (*depthai.SystemInformation* property), [192](#)  
[leonCssMemoryUsage\(\)](#) (*depthai.RawSystemInformation* property), [178](#)  
[leonCssMemoryUsage\(\)](#) (*depthai.SystemInformation* property), [192](#)  
[leonMssCpuUsage\(\)](#) (*depthai.RawSystemInformation* property), [178](#)  
[leonMssCpuUsage\(\)](#) (*depthai.SystemInformation* property), [192](#)  
[leonMssMemoryUsage\(\)](#) (*depthai.RawSystemInformation* property), [178](#)  
[leonMssMemoryUsage\(\)](#) (*depthai.SystemInformation* property), [192](#)  
[leonOsFrequencyHz\(\)](#) (*depthai.GlobalProperties* property), [136](#)  
[leonRtFrequencyHz\(\)](#) (*depthai.GlobalProperties* property), [136](#)  
[link\(\)](#) (*depthai.Node.Output* method), [157](#)  
[link\(\)](#) (*depthai.Pipeline* method), [162](#)  
[loadCalibrationData\(\)](#) (*depthai.StereoDepth* method), [188](#)  
[loadCalibrationFile\(\)](#) (*depthai.StereoDepth* method), [189](#)  
[lockAspectRatioFill\(\)](#) (*depthai.RawImageManipConfig.ResizeConfig* property), [172](#)  
[LogLevel](#) (class in *depthai*), [146](#)  
[lowerThreshold\(\)](#) (*depthai.SpatialLocationCalculatorConfigThreshold* property), [186](#)  
[LUT16](#) (*depthai.ImgFrame.Type* attribute), [145](#)  
[LUT16](#) (*depthai.RawImgFrame.Type* attribute), [176](#)  
[LUT2](#) (*depthai.ImgFrame.Type* attribute), [145](#)  
[LUT2](#) (*depthai.RawImgFrame.Type* attribute), [176](#)  
[LUT4](#) (*depthai.ImgFrame.Type* attribute), [145](#)  
[LUT4](#) (*depthai.RawImgFrame.Type* attribute), [176](#)

## L

[label\(\)](#) (*depthai.ImgDetection* property), [141](#)  
[LANDSCAPE](#) (*depthai.CameraControl.SceneMode* attribute), [110](#)  
[LANDSCAPE](#) (*depthai.RawCameraControl.SceneMode* attribute), [169](#)  
[LEFT](#) (*depthai.CameraBoardSocket* attribute), [104](#)  
[left\(\)](#) (*depthai.StereoDepth* property), [188](#)  
[lensPosition\(\)](#) (*depthai.RawCameraControl* property), [169](#)  
[leonCssCpuUsage\(\)](#) (*depthai.RawSystemInformation* property), [178](#)

## M

[MACRO](#) (*depthai.CameraControl.AutoFocusMode* attribute), [106](#)  
[MACRO](#) (*depthai.RawCameraControl.AutoFocusMode* attribute), [166](#)  
[MAINS\\_50\\_HZ](#) (*depthai.CameraControl.AntiBandingMode* attribute), [105](#)  
[MAINS\\_50\\_HZ](#) (*depthai.RawCameraControl.AntiBandingMode* attribute), [165](#)  
[MAINS\\_60\\_HZ](#) (*depthai.CameraControl.AntiBandingMode* attribute), [106](#)  
[MAINS\\_60\\_HZ](#) (*depthai.RawCameraControl.AntiBandingMode* attribute), [165](#)

maxBitrate() (*depthai.VideoEncoderProperties* property), 201  
 median() (*depthai.StereoDepthProperties* property), 191  
 MEDIAN\_OFF (*depthai.StereoDepthProperties.MedianFilter* attribute), 191  
 MemoryInfo (class in *depthai*), 147  
 MJPEG (*depthai.VideoEncoderProperties.Profile* attribute), 200  
 MobileNetDetectionNetwork (class in *depthai*), 148  
 MobileNetSpatialDetectionNetwork (class in *depthai*), 148  
 module  
     *depthai*, 99  
 MONO (*depthai.CameraControl.EffectMode* attribute), 108  
 MONO (*depthai.RawCameraControl.EffectMode* attribute), 167  
 MonoCamera (class in *depthai*), 148  
 MonoCameraProperties (class in *depthai*), 150  
 MonoCameraProperties.SensorResolution (class in *depthai*), 150  
 mss() (*depthai.ChipTemperature* property), 113  
 msTime() (*depthai.CpuUsage* property), 121  
**N**  
 name() (*depthai.CameraBoardSocket* property), 104  
 name() (*depthai.CameraControl.AntiBandingMode* property), 106  
 name() (*depthai.CameraControl.AutoFocusMode* property), 106  
 name() (*depthai.CameraControl.AutoWhiteBalanceMode* property), 107  
 name() (*depthai.CameraControl.EffectMode* property), 108  
 name() (*depthai.CameraControl.SceneMode* property), 110  
 name() (*depthai.CameraImageOrientation* property), 113  
 name() (*depthai.ColorCameraProperties.ColorOrder* property), 119  
 name() (*depthai.ColorCameraProperties.SensorResolution* property), 120  
 name() (*depthai.DeviceDesc* property), 135  
 name() (*depthai.ImgFrame.Type* property), 145  
 name() (*depthai.LogLevel* property), 147  
 name() (*depthai.MonoCameraProperties.SensorResolution* property), 151  
 name() (*depthai.OpenVINO.Version* property), 159  
 name() (*depthai.RawCameraControl.AntiBandingMode* property), 165  
 name() (*depthai.RawCameraControl.AutoFocusMode* property), 166  
 name() (*depthai.RawCameraControl.AutoWhiteBalanceMode* property), 167  
 name() (*depthai.RawCameraControl.EffectMode* property), 168  
 name() (*depthai.RawCameraControl.SceneMode* property), 169  
 name() (*depthai.RawImgFrame.Type* property), 176  
 name() (*depthai.StereoDepthProperties.MedianFilter* property), 191  
 name() (*depthai.TensorInfo* property), 195  
 name() (*depthai.TensorInfo.DataType* property), 194  
 name() (*depthai.TensorInfo.StorageOrder* property), 195  
 name() (*depthai.VideoEncoderProperties.Profile* property), 200  
 name() (*depthai.VideoEncoderProperties.RateControlMode* property), 201  
 name() (*depthai.XLinkDeviceState* property), 202  
 name() (*depthai.XLinkPlatform* property), 205  
 name() (*depthai.XLinkProtocol* property), 205  
 NC (*depthai.TensorInfo.StorageOrder* attribute), 195  
 NCHW (*depthai.TensorInfo.StorageOrder* attribute), 195  
 NEGATIVE (*depthai.CameraControl.EffectMode* attribute), 108  
 NEGATIVE (*depthai.RawCameraControl.EffectMode* attribute), 167  
 NeuralNetwork (class in *depthai*), 153  
 NeuralNetworkProperties (class in *depthai*), 155  
 NHCW (*depthai.TensorInfo.StorageOrder* attribute), 195  
 NHWC (*depthai.TensorInfo.StorageOrder* attribute), 195  
 NIGHT (*depthai.CameraControl.SceneMode* attribute), 110  
 NIGHT (*depthai.RawCameraControl.SceneMode* attribute), 169  
 NIGHT\_PORTRAIT (*depthai.CameraControl.SceneMode* attribute), 110  
 NIGHT\_PORTRAIT (*depthai.RawCameraControl.SceneMode* attribute), 169  
 NNData (class in *depthai*), 151  
 nnFamily() (*depthai.DetectionNetworkProperties* property), 125  
 Node (class in *depthai*), 155  
 Node.Connection (class in *depthai*), 156  
 Node.Id (class in *depthai*), 156  
 Node.Input (class in *depthai*), 157  
 Node.Output (class in *depthai*), 157  
 NONE (*depthai.ImgFrame.Type* attribute), 145  
 NONE (*depthai.RawImgFrame.Type* attribute), 176  
 NORMAL (*depthai.CameraImageOrientation* attribute), 113  
 normalize() (*depthai.Rect* method), 179  
 normalizedCoords() (*depthai.RawImageManipConfig.CropConfig* property), 170

normalizedCoords() (*depthai.RawImageManipConfig.ResizeConfig property*), 172

nsec() (*depthai.Timestamp property*), 196

numBFrames() (*depthai.VideoEncoderProperties property*), 201

numDimensions() (*depthai.TensorInfo property*), 195

numFrames() (*depthai.NeuralNetworkProperties property*), 155

numFramesPool() (*depthai.VideoEncoderProperties property*), 201

numNCEPerThread() (*depthai.NeuralNetworkProperties property*), 155

numThreads() (*depthai.NeuralNetworkProperties property*), 155

NV12 (*depthai.ImgFrame.Type attribute*), 145

NV12 (*depthai.RawImgFrame.Type attribute*), 176

NV21 (*depthai.ImgFrame.Type attribute*), 145

NV21 (*depthai.RawImgFrame.Type attribute*), 176

## O

OFF (*depthai.CameraControl.AntiBandingMode attribute*), 106

OFF (*depthai.CameraControl.AutoFocusMode attribute*), 106

OFF (*depthai.CameraControl.AutoWhiteBalanceMode attribute*), 107

OFF (*depthai.CameraControl.EffectMode attribute*), 108

OFF (*depthai.LogLevel attribute*), 147

OFF (*depthai.RawCameraControl.AntiBandingMode attribute*), 165

OFF (*depthai.RawCameraControl.AutoFocusMode attribute*), 166

OFF (*depthai.RawCameraControl.AutoWhiteBalanceMode attribute*), 167

OFF (*depthai.RawCameraControl.EffectMode attribute*), 167

offset() (*depthai.TensorInfo property*), 195

OpenVINO (*class in depthai*), 158

OpenVINO.Version (*class in depthai*), 159

order() (*depthai.TensorInfo property*), 195

out() (*depthai.DetectionNetwork property*), 125

out() (*depthai.ImageManip property*), 137

out() (*depthai.MonoCamera property*), 150

out() (*depthai.NeuralNetwork property*), 154

out() (*depthai.SpatialDetectionNetwork property*), 182

out() (*depthai.SpatialLocationCalculator property*), 184

out() (*depthai.SystemLogger property*), 192

out() (*depthai.XLinkIn property*), 203

outputId() (*depthai.Node.Connection property*), 156

outputName() (*depthai.Node.Connection property*), 156

## P

p1Offset() (*depthai.ImgFrame.Specs property*), 142

p1Offset() (*depthai.RawImgFrame.Specs property*), 174

p2Offset() (*depthai.ImgFrame.Specs property*), 142

p2Offset() (*depthai.RawImgFrame.Specs property*), 174

p3Offset() (*depthai.ImgFrame.Specs property*), 142

p3Offset() (*depthai.RawImgFrame.Specs property*), 174

PACK10 (*depthai.ImgFrame.Type attribute*), 145

PACK10 (*depthai.RawImgFrame.Type attribute*), 176

PACK12 (*depthai.ImgFrame.Type attribute*), 145

PACK12 (*depthai.RawImgFrame.Type attribute*), 176

parseVersionName() (*depthai.OpenVINO static method*), 160

PARTY (*depthai.CameraControl.SceneMode attribute*), 110

PARTY (*depthai.RawCameraControl.SceneMode attribute*), 169

passthrough() (*depthai.DetectionNetwork property*), 125

passthrough() (*depthai.NeuralNetwork property*), 154

passthrough() (*depthai.SpatialDetectionNetwork property*), 182

passthroughDepth() (*depthai.SpatialDetectionNetwork property*), 182

passthroughDepth() (*depthai.SpatialLocationCalculator property*), 184

Pipeline (*class in depthai*), 160

pipelineName() (*depthai.GlobalProperties property*), 136

pipelineVersion() (*depthai.GlobalProperties property*), 136

platform() (*depthai.DeviceDesc property*), 135

Point2f (*class in depthai*), 163

Point3f (*class in depthai*), 163

PORTRAIT (*depthai.CameraControl.SceneMode attribute*), 110

PORTRAIT (*depthai.RawCameraControl.SceneMode attribute*), 169

POSTERIZE (*depthai.CameraControl.EffectMode attribute*), 108

POSTERIZE (*depthai.RawCameraControl.EffectMode attribute*), 168

preview() (*depthai.ColorCamera property*), 117

previewHeight() (*depthai.ColorCameraProperties property*), 120

previewWidth() (*depthai.ColorCameraProperties property*), 120



[profile\(\)](#) (*depthai.VideoEncoderProperties* property), 201  
[Properties](#) (*depthai.ColorCamera* attribute), 115  
[Properties](#) (*depthai.DetectionNetwork* attribute), 124  
[Properties](#) (*depthai.MonoCamera* attribute), 149  
[Properties](#) (*depthai.NeuralNetwork* attribute), 154  
[Properties](#) (*depthai.SpatialDetectionNetwork* attribute), 181  
[Properties](#) (*depthai.SpatialLocationCalculator* attribute), 184  
[Properties](#) (*depthai.StereoDepth* attribute), 188  
[Properties](#) (*depthai.VideoEncoder* attribute), 197  
[protocol\(\)](#) (*depthai.DeviceDesc* property), 135

## Q

[quality\(\)](#) (*depthai.VideoEncoderProperties* property), 201

## R

[rateCtrlMode\(\)](#) (*depthai.VideoEncoderProperties* property), 201  
[rateHz\(\)](#) (*depthai.SystemLoggerProperties* property), 193  
[RAW10](#) (*depthai.ImgFrame.Type* attribute), 145  
[RAW10](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RAW12](#) (*depthai.ImgFrame.Type* attribute), 145  
[RAW12](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RAW14](#) (*depthai.ImgFrame.Type* attribute), 145  
[RAW14](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RAW16](#) (*depthai.ImgFrame.Type* attribute), 145  
[RAW16](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RAW8](#) (*depthai.ImgFrame.Type* attribute), 145  
[RAW8](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RawBuffer](#) (class in *depthai*), 164  
[RawCameraControl](#) (class in *depthai*), 164  
[RawCameraControl.AntiBandingMode](#) (class in *depthai*), 164  
[RawCameraControl.AutoFocusMode](#) (class in *depthai*), 165  
[RawCameraControl.AutoWhiteBalanceMode](#) (class in *depthai*), 166  
[RawCameraControl.EffectMode](#) (class in *depthai*), 167  
[RawCameraControl.SceneMode](#) (class in *depthai*), 168  
[RawImageManipConfig](#) (class in *depthai*), 169  
[RawImageManipConfig.CropConfig](#) (class in *depthai*), 170  
[RawImageManipConfig.CropRect](#) (class in *depthai*), 170  
[RawImageManipConfig.FormatConfig](#) (class in *depthai*), 171  
[RawImageManipConfig.ResizeConfig](#) (class in *depthai*), 171

[RawImgDetections](#) (class in *depthai*), 172  
[RawImgFrame](#) (class in *depthai*), 173  
[RawImgFrame.Specs](#) (class in *depthai*), 173  
[RawImgFrame.Type](#) (class in *depthai*), 174  
[RawNNData](#) (class in *depthai*), 177  
[RawSpatialImgDetections](#) (class in *depthai*), 177  
[RawSystemInformation](#) (class in *depthai*), 177  
[Rect](#) (class in *depthai*), 178  
[rectifiedLeft\(\)](#) (*depthai.StereoDepth* property), 189  
[rectifiedRight\(\)](#) (*depthai.StereoDepth* property), 189  
[rectifyEdgeFillColor\(\)](#) (*depthai.StereoDepthProperties* property), 191  
[rectifyMirrorFrame\(\)](#) (*depthai.StereoDepthProperties* property), 191  
[remaining\(\)](#) (*depthai.MemoryInfo* property), 147  
[remove\(\)](#) (*depthai.AssetManager* method), 102  
[remove\(\)](#) (*depthai.Pipeline* method), 162  
[removeCallback\(\)](#) (*depthai.DataOutputQueue* method), 123  
[removeLogCallback\(\)](#) (*depthai.Device* method), 132  
[resizeConfig\(\)](#) (*depthai.RawImageManipConfig* property), 172  
[resolution\(\)](#) (*depthai.ColorCameraProperties* property), 120  
[resolution\(\)](#) (*depthai.MonoCameraProperties* property), 151  
[RGB](#) (*depthai.CameraBoardSocket* attribute), 104  
[RGB](#) (*depthai.ColorCameraProperties.ColorOrder* attribute), 119  
[RGB161616](#) (*depthai.ImgFrame.Type* attribute), 145  
[RGB161616](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RGB888i](#) (*depthai.ImgFrame.Type* attribute), 145  
[RGB888i](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RGB888p](#) (*depthai.ImgFrame.Type* attribute), 145  
[RGB888p](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RGBA8888](#) (*depthai.ImgFrame.Type* attribute), 145  
[RGBA8888](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RGBF16F16F16i](#) (*depthai.ImgFrame.Type* attribute), 145  
[RGBF16F16F16i](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RGBF16F16F16p](#) (*depthai.ImgFrame.Type* attribute), 145  
[RGBF16F16F16p](#) (*depthai.RawImgFrame.Type* attribute), 176  
[RIGHT](#) (*depthai.CameraBoardSocket* attribute), 104  
[right\(\)](#) (*depthai.StereoDepth* property), 189  
[roi\(\)](#) (*depthai.SpatialLocationCalculatorConfigData*

property), 185  
 roiConfig() (*depthai.SpatialLocationCalculatorProperties*  
 property), 186  
 ROTATE\_180\_DEG (*depthai.CameraImageOrientation*  
 attribute), 113  
 RotatedRect (class in *depthai*), 179  
 rotationAngleDeg()  
 (*depthai.RawImageManipConfig.ResizeConfig*  
 property), 172

## S

saveDepthaiApplicationPackage()  
 (*depthai.DeviceBootloader* static method),  
 134  
 sec() (*depthai.Timestamp* property), 196  
 send() (*depthai.DataInputQueue* method), 121  
 sensorCenterCrop() (*depthai.ColorCamera*  
 method), 117  
 sensorCropX() (*depthai.ColorCameraProperties*  
 property), 120  
 sensorCropY() (*depthai.ColorCameraProperties*  
 property), 120  
 SEPIA (*depthai.CameraControl.EffectMode* attribute),  
 108  
 SEPIA (*depthai.RawCameraControl.EffectMode* at-  
 tribute), 168  
 sequenceNum() (*depthai.RawImgFrame* property),  
 177  
 set() (*depthai.AssetManager* method), 102  
 setAnchorMasks() (*depthai.YoloDetectionNetwork*  
 method), 206  
 setAnchorMasks() (*depthai.YoloSpatialDetectionNetwork*  
 method), 206  
 setAnchors() (*depthai.YoloDetectionNetwork*  
 method), 206  
 setAnchors() (*depthai.YoloSpatialDetectionNetwork*  
 method), 206  
 setAntiBandingMode() (*depthai.CameraControl*  
 method), 110  
 setAutoExposureCompensation()  
 (*depthai.CameraControl* method), 110  
 setAutoExposureEnable()  
 (*depthai.CameraControl* method), 110  
 setAutoExposureLock() (*depthai.CameraControl*  
 method), 110  
 setAutoExposureRegion()  
 (*depthai.CameraControl* method), 111  
 setAutoFocusMode() (*depthai.CameraControl*  
 method), 111  
 setAutoFocusRegion() (*depthai.CameraControl*  
 method), 111  
 setAutoFocusTrigger() (*depthai.CameraControl*  
 method), 111

setAutoWhiteBalanceLock()  
 (*depthai.CameraControl* method), 111  
 setAutoWhiteBalanceMode()  
 (*depthai.CameraControl* method), 111  
 setBitrate() (*depthai.VideoEncoder* method), 198  
 setBitrateKbps() (*depthai.VideoEncoder* method),  
 198  
 setBlobPath() (*depthai.NeuralNetwork* method),  
 155  
 setBlocking() (*depthai.DataInputQueue* method),  
 122  
 setBlocking() (*depthai.DataOutputQueue* method),  
 123  
 setBlocking() (*depthai.Node.Input* method), 157  
 setBoardSocket() (*depthai.ColorCamera* method),  
 117  
 setBoardSocket() (*depthai.MonoCamera* method),  
 150  
 setBoundingBoxScaleFactor()  
 (*depthai.SpatialDetectionNetwork* method),  
 182  
 setBrightness() (*depthai.CameraControl* method),  
 111  
 setBusId() (*depthai.SPIOut* method), 180  
 setCamId() (*depthai.ColorCamera* method), 117  
 setCamId() (*depthai.MonoCamera* method), 150  
 setCaptureStill() (*depthai.CameraControl*  
 method), 111  
 setCategory() (*depthai.ImgFrame* method), 146  
 setCenterCrop() (*depthai.ImageManip* method),  
 137  
 setCenterCrop() (*depthai.ImageManipConfig*  
 method), 139  
 setChromaDenoise() (*depthai.CameraControl*  
 method), 111  
 setColorOrder() (*depthai.ColorCamera* method),  
 117  
 setConfidenceThreshold()  
 (*depthai.DetectionNetwork* method), 125  
 setConfidenceThreshold()  
 (*depthai.StereoDepth* method), 189  
 setContrast() (*depthai.CameraControl* method),  
 111  
 setCoordinateSize()  
 (*depthai.YoloDetectionNetwork* method),  
 206  
 setCoordinateSize()  
 (*depthai.YoloSpatialDetectionNetwork*  
 method), 206  
 setCropRect() (*depthai.ImageManip* method), 137  
 setCropRotatedRect()  
 (*depthai.ImageManipConfig* method), 139  
 setData() (*depthai.Buffer* method), 103  
 setDefaultProfilePreset()

(*depthai.VideoEncoder method*), 198  
 setDepthLowerThreshold() (*depthai.SpatialDetectionNetwork method*), 182  
 setDepthUpperThreshold() (*depthai.SpatialDetectionNetwork method*), 182  
 setEffectMode() (*depthai.CameraControl method*), 111  
 setEmptyCalibration() (*depthai.StereoDepth method*), 189  
 setExtendedDisparity() (*depthai.StereoDepth method*), 189  
 setFp16() (*depthai.ColorCamera method*), 118  
 setFps() (*depthai.ColorCamera method*), 118  
 setFps() (*depthai.MonoCamera method*), 150  
 setFpsLimit() (*depthai.XLinkOut method*), 204  
 setFrame() (*depthai.ImgFrame method*), 146  
 setFrameRate() (*depthai.VideoEncoder method*), 198  
 setFrameType() (*depthai.ImageManip method*), 137  
 setFrameType() (*depthai.ImageManipConfig method*), 139  
 setHeight() (*depthai.ImgFrame method*), 146  
 setHorizontalFlip() (*depthai.ImageManip method*), 137  
 setHorizontalFlip() (*depthai.ImageManipConfig method*), 139  
 setImageOrientation() (*depthai.ColorCamera method*), 118  
 setImageOrientation() (*depthai.MonoCamera method*), 150  
 setInputResolution() (*depthai.StereoDepth method*), 189  
 setInstanceNum() (*depthai.ImgFrame method*), 146  
 setInterleaved() (*depthai.ColorCamera method*), 118  
 setIouThreshold() (*depthai.YoloDetectionNetwork method*), 206  
 setIouThreshold() (*depthai.YoloSpatialDetectionNetwork method*), 206  
 setKeepAspectRatio() (*depthai.ImageManip method*), 137  
 setKeepAspectRatio() (*depthai.ImageManipConfig method*), 139  
 setKeyframeFrequency() (*depthai.VideoEncoder method*), 199  
 setLayer() (*depthai.NNData method*), 153  
 setLeftRightCheck() (*depthai.StereoDepth method*), 189  
 setLogLevel() (*depthai.Device method*), 132  
 setLogOutputLevel() (*depthai.Device method*), 132  
 setLumaDenoise() (*depthai.CameraControl method*), 111  
 setManualExposure() (*depthai.CameraControl method*), 112  
 setManualFocus() (*depthai.CameraControl method*), 112  
 setMaxDataSize() (*depthai.XLinkIn method*), 203  
 setMaxOutputFrameSize() (*depthai.ImageManip method*), 137  
 setMaxSize() (*depthai.DataInputQueue method*), 122  
 setMaxSize() (*depthai.DataOutputQueue method*), 123  
 setMedianFilter() (*depthai.StereoDepth method*), 189  
 setMetadataOnly() (*depthai.XLinkOut method*), 204  
 setNoiseReductionStrength() (*depthai.CameraControl method*), 112  
 setNumBframes() (*depthai.VideoEncoder method*), 199  
 setNumClasses() (*depthai.YoloDetectionNetwork method*), 206  
 setNumClasses() (*depthai.YoloSpatialDetectionNetwork method*), 206  
 setNumFrames() (*depthai.XLinkIn method*), 203  
 setNumFramesPool() (*depthai.ImageManip method*), 137  
 setNumFramesPool() (*depthai.VideoEncoder method*), 199  
 setNumInferenceThreads() (*depthai.NeuralNetwork method*), 155  
 setNumNCEPerInferenceThread() (*depthai.NeuralNetwork method*), 155  
 setNumPoolFrames() (*depthai.NeuralNetwork method*), 155  
 setOpenVINOVersion() (*depthai.Pipeline method*), 162  
 setOutputDepth() (*depthai.StereoDepth method*), 189  
 setOutputRectified() (*depthai.StereoDepth method*), 189  
 setPreviewKeepAspectRatio() (*depthai.ColorCamera method*), 118  
 setPreviewSize() (*depthai.ColorCamera method*), 118  
 setProfile() (*depthai.VideoEncoder method*), 199  
 setQuality() (*depthai.VideoEncoder method*), 199  
 setQueueSize() (*depthai.Node.Input method*), 157  
 setRate() (*depthai.SystemLogger method*), 192  
 setRateControlMode() (*depthai.VideoEncoder method*), 199  
 setRectifyEdgeFillColor()

- [\(depthai.StereoDepth method\), 189](#)
- [setRectifyMirrorFrame\(\) \(depthai.StereoDepth method\), 189](#)
- [setResize\(\) \(depthai.ImageManip method\), 137](#)
- [setResize\(\) \(depthai.ImageManipConfig method\), 139](#)
- [setResizeThumbnail\(\) \(depthai.ImageManip method\), 137](#)
- [setResizeThumbnail\(\) \(depthai.ImageManipConfig method\), 139](#)
- [setResolution\(\) \(depthai.ColorCamera method\), 118](#)
- [setResolution\(\) \(depthai.MonoCamera method\), 150](#)
- [setReusePreviousImage\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setROIs\(\) \(depthai.SpatialLocationCalculatorConfig method\), 185](#)
- [setRotationDegrees\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setRotationRadians\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setSaturation\(\) \(depthai.CameraControl method\), 112](#)
- [setSceneMode\(\) \(depthai.CameraControl method\), 112](#)
- [setSensorCrop\(\) \(depthai.ColorCamera method\), 118](#)
- [setSequenceNum\(\) \(depthai.ImgFrame method\), 146](#)
- [setSharpness\(\) \(depthai.CameraControl method\), 112](#)
- [setSkipCurrentImage\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setStartStreaming\(\) \(depthai.CameraControl method\), 112](#)
- [setStillSize\(\) \(depthai.ColorCamera method\), 118](#)
- [setStopStreaming\(\) \(depthai.CameraControl method\), 112](#)
- [setStreamName\(\) \(depthai.SPIOut method\), 180](#)
- [setStreamName\(\) \(depthai.XLinkIn method\), 203](#)
- [setStreamName\(\) \(depthai.XLinkOut method\), 204](#)
- [setSubpixel\(\) \(depthai.StereoDepth method\), 189](#)
- [setSystemInformationLoggingRate\(\) \(depthai.Device method\), 132](#)
- [setTimestamp\(\) \(depthai.ImgFrame method\), 146](#)
- [setType\(\) \(depthai.ImgFrame method\), 146](#)
- [setVideoSize\(\) \(depthai.ColorCamera method\), 118](#)
- [setWaitForConfigInput\(\) \(depthai.ColorCamera method\), 118](#)
- [setWaitForConfigInput\(\) \(depthai.ImageManip method\), 137](#)
- [setWaitForConfigInput\(\)](#)
- [\(depthai.SpatialLocationCalculator method\), 184](#)
- [setWarpBorderFillColor\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setWarpBorderReplicatePixels\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setWarpTransformFourPoints\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setWarpTransformMatrix3x3\(\) \(depthai.ImageManipConfig method\), 140](#)
- [setWidth\(\) \(depthai.ImgFrame method\), 146](#)
- [SHADE \(depthai.CameraControl.AutoWhiteBalanceMode attribute\), 107](#)
- [SHADE \(depthai.RawCameraControl.AutoWhiteBalanceMode attribute\), 167](#)
- [size\(\) \(depthai.AssetManager method\), 103](#)
- [size\(\) \(depthai.Rect method\), 179](#)
- [size\(\) \(depthai.RotatedRect property\), 180](#)
- [Size2f \(class in depthai\), 180](#)
- [SNOW \(depthai.CameraControl.SceneMode attribute\), 110](#)
- [SNOW \(depthai.RawCameraControl.SceneMode attribute\), 169](#)
- [SOLARIZE \(depthai.CameraControl.EffectMode attribute\), 108](#)
- [SOLARIZE \(depthai.RawCameraControl.EffectMode attribute\), 168](#)
- [spatialCoordinates\(\) \(depthai.SpatialImgDetection property\), 183](#)
- [spatialCoordinates\(\) \(depthai.SpatialLocations property\), 187](#)
- [SpatialDetectionNetwork \(class in depthai\), 181](#)
- [SpatialDetectionNetworkProperties \(class in depthai\), 182](#)
- [SpatialImgDetection \(class in depthai\), 183](#)
- [SpatialImgDetections \(class in depthai\), 183](#)
- [SpatialLocationCalculator \(class in depthai\), 183](#)
- [SpatialLocationCalculatorConfig \(class in depthai\), 184](#)
- [SpatialLocationCalculatorConfigData \(class in depthai\), 185](#)
- [SpatialLocationCalculatorConfigThresholds \(class in depthai\), 185](#)
- [SpatialLocationCalculatorData \(class in depthai\), 186](#)
- [SpatialLocationCalculatorProperties \(class in depthai\), 186](#)
- [SpatialLocations \(class in depthai\), 186](#)
- [SPIOut \(class in depthai\), 180](#)
- [SPORTS \(depthai.CameraControl.SceneMode attribute\), 110](#)
- [SPORTS \(depthai.RawCameraControl.SceneMode at-](#)



tribute), 169  
 startPipeline() (depthai.Device method), 132  
 state() (depthai.DeviceInfo property), 135  
 STEADYPHOTO (depthai.CameraControl.SceneMode attribute), 110  
 STEADYPHOTO (depthai.RawCameraControl.SceneMode attribute), 169  
 StereoDepth (class in depthai), 187  
 StereoDepthProperties (class in depthai), 190  
 StereoDepthProperties.MedianFilter (class in depthai), 190  
 still() (depthai.ColorCamera property), 118  
 stillHeight() (depthai.ColorCameraProperties property), 120  
 stillWidth() (depthai.ColorCameraProperties property), 120  
 stride() (depthai.ImgFrame.Specs property), 142  
 stride() (depthai.RawImgFrame.Specs property), 174  
 strides() (depthai.TensorInfo property), 196  
 SUNSET (depthai.CameraControl.SceneMode attribute), 110  
 SUNSET (depthai.RawCameraControl.SceneMode attribute), 169  
 syncedLeft() (depthai.StereoDepth property), 190  
 syncedRight() (depthai.StereoDepth property), 190  
 SystemInformation (class in depthai), 191  
 SystemLogger (class in depthai), 192  
 SystemLoggerProperties (class in depthai), 192

## T

TensorInfo (class in depthai), 193  
 TensorInfo.DataType (class in depthai), 193  
 TensorInfo.StorageOrder (class in depthai), 194  
 tensors() (depthai.RawNNData property), 177  
 THE\_1080\_P (depthai.ColorCameraProperties.SensorResolution attribute), 120  
 THE\_12\_MP (depthai.ColorCameraProperties.SensorResolution attribute), 120  
 THE\_400\_P (depthai.MonoCameraProperties.SensorResolution attribute), 151  
 THE\_4\_K (depthai.ColorCameraProperties.SensorResolution attribute), 120  
 THE\_720\_P (depthai.MonoCameraProperties.SensorResolution attribute), 151  
 THE\_800\_P (depthai.MonoCameraProperties.SensorResolution attribute), 151  
 THEATRE (depthai.CameraControl.SceneMode attribute), 110  
 THEATRE (depthai.RawCameraControl.SceneMode attribute), 169  
 Timestamp (class in depthai), 196  
 topLeft() (depthai.Rect method), 179  
 total() (depthai.MemoryInfo property), 148  
 TRACE (depthai.LogLevel attribute), 147

tryGet() (depthai.DataOutputQueue method), 124  
 tryGetAll() (depthai.DataOutputQueue method), 124  
 ts() (depthai.RawImgFrame property), 177  
 TWILIGHT (depthai.CameraControl.AutoWhiteBalanceMode attribute), 107  
 TWILIGHT (depthai.RawCameraControl.AutoWhiteBalanceMode attribute), 167  
 type() (depthai.ImgFrame.Specs property), 142  
 type() (depthai.RawImageManipConfig.FormatConfig property), 171  
 type() (depthai.RawImgFrame.Specs property), 174

## U

U8F (depthai.TensorInfo.DataType attribute), 194  
 unlink() (depthai.Node.Output method), 158  
 unlink() (depthai.Pipeline method), 162  
 UNSUPPORTED (depthai.CameraControl.SceneMode attribute), 110  
 UNSUPPORTED (depthai.RawCameraControl.SceneMode attribute), 169  
 upa() (depthai.ChipTemperature property), 113  
 upperThreshold() (depthai.SpatialLocationCalculatorConfigThreshold property), 186  
 used() (depthai.MemoryInfo property), 148

## V

value() (depthai.CameraBoardSocket property), 104  
 value() (depthai.CameraControl.AntiBandingMode property), 106  
 value() (depthai.CameraControl.AutoFocusMode property), 106  
 value() (depthai.CameraControl.AutoWhiteBalanceMode property), 107  
 value() (depthai.CameraControl.EffectMode property), 108  
 value() (depthai.CameraControl.SceneMode property), 110  
 value() (depthai.CameraImageOrientation property), 113  
 value() (depthai.ColorCameraProperties.ColorOrder property), 119  
 value() (depthai.ColorCameraProperties.SensorResolution property), 120  
 value() (depthai.ImgFrame.Type property), 145  
 value() (depthai.LogLevel property), 147  
 value() (depthai.MonoCameraProperties.SensorResolution property), 151  
 value() (depthai.OpenVINO.Version property), 160  
 value() (depthai.RawCameraControl.AntiBandingMode property), 165  
 value() (depthai.RawCameraControl.AutoFocusMode property), 166

- `value()` (*depthai.RawCameraControl.AutoWhiteBalanceMode* property), 167
- `value()` (*depthai.RawCameraControl.EffectMode* property), 168
- `value()` (*depthai.RawCameraControl.SceneMode* property), 169
- `value()` (*depthai.RawImgFrame.Type* property), 177
- `value()` (*depthai.StereoDepthProperties.MedianFilter* property), 191
- `value()` (*depthai.TensorInfo.DataType* property), 194
- `value()` (*depthai.TensorInfo.StorageOrder* property), 195
- `value()` (*depthai.VideoEncoderProperties.Profile* property), 200
- `value()` (*depthai.VideoEncoderProperties.RateControlMode* property), 201
- `value()` (*depthai.XLinkDeviceState* property), 202
- `value()` (*depthai.XLinkPlatform* property), 205
- `value()` (*depthai.XLinkProtocol* property), 205
- `VBR` (*depthai.VideoEncoderProperties.RateControlMode* attribute), 201
- `VERSION_2020_1` (*depthai.OpenVINO* attribute), 159
- `VERSION_2020_1` (*depthai.OpenVINO.Version* attribute), 159
- `VERSION_2020_2` (*depthai.OpenVINO* attribute), 159
- `VERSION_2020_2` (*depthai.OpenVINO.Version* attribute), 159
- `VERSION_2020_3` (*depthai.OpenVINO* attribute), 159
- `VERSION_2020_3` (*depthai.OpenVINO.Version* attribute), 159
- `VERSION_2020_4` (*depthai.OpenVINO* attribute), 159
- `VERSION_2020_4` (*depthai.OpenVINO.Version* attribute), 159
- `VERSION_2021_1` (*depthai.OpenVINO* attribute), 159
- `VERSION_2021_1` (*depthai.OpenVINO.Version* attribute), 159
- `VERSION_2021_2` (*depthai.OpenVINO* attribute), 159
- `VERSION_2021_2` (*depthai.OpenVINO.Version* attribute), 159
- `VERTICAL_FLIP` (*depthai.CameraImageOrientation* attribute), 113
- `video()` (*depthai.ColorCamera* property), 118
- `VideoEncoder` (class in *depthai*), 196
- `VideoEncoderProperties` (class in *depthai*), 199
- `VideoEncoderProperties.Profile` (class in *depthai*), 200
- `VideoEncoderProperties.RateControlMode` (class in *depthai*), 200
- `videoHeight()` (*depthai.ColorCameraProperties* property), 120
- `videoWidth()` (*depthai.ColorCameraProperties* property), 121
- `W` (*depthai.TensorInfo.StorageOrder* attribute), 195
- `WARM_FLUORESCENT` (*depthai.CameraControl.AutoWhiteBalanceMode* attribute), 107
- `WARM_FLUORESCENT` (*depthai.RawCameraControl.AutoWhiteBalanceMode* attribute), 167
- `WARN` (*depthai.LogLevel* attribute), 147
- `warpBorderReplicate()` (*depthai.RawImageManipConfig.ResizeConfig* property), 172
- `warpFourPoints()` (*depthai.RawImageManipConfig.ResizeConfig* property), 172
- `warpMatrix3x3()` (*depthai.RawImageManipConfig.ResizeConfig* property), 172
- `WCH` (*depthai.TensorInfo.StorageOrder* attribute), 195
- `WHC` (*depthai.TensorInfo.StorageOrder* attribute), 195
- `WHITEBOARD` (*depthai.CameraControl.EffectMode* attribute), 108
- `WHITEBOARD` (*depthai.RawCameraControl.EffectMode* attribute), 168
- `width()` (*depthai.ImgFrame.Specs* property), 142
- `width()` (*depthai.RawImageManipConfig.ResizeConfig* property), 172
- `width()` (*depthai.RawImgFrame.Specs* property), 174
- `width()` (*depthai.Rect* property), 179
- `width()` (*depthai.Size2f* property), 181
- `width()` (*depthai.StereoDepthProperties* property), 191
- `width()` (*depthai.VideoEncoderProperties* property), 201
- `widthHeightAspectRatio()` (*depthai.RawImageManipConfig.CropConfig* property), 170
- X**
- `x()` (*depthai.Point2f* property), 163
- `x()` (*depthai.Point3f* property), 163
- `x()` (*depthai.Rect* property), 179
- `X_LINK_ANY_PLATFORM` (*depthai.XLinkPlatform* attribute), 204
- `X_LINK_ANY_PROTOCOL` (*depthai.XLinkProtocol* attribute), 205
- `X_LINK_ANY_STATE` (*depthai.XLinkDeviceState* attribute), 202
- `X_LINK_BOOTED` (*depthai.XLinkDeviceState* attribute), 202
- `X_LINK_BOOTLOADER` (*depthai.XLinkDeviceState* attribute), 202
- `X_LINK_IPC` (*depthai.XLinkProtocol* attribute), 205
- `X_LINK_MYRIAD_2` (*depthai.XLinkPlatform* attribute), 204
- `X_LINK_MYRIAD_X` (*depthai.XLinkPlatform* attribute), 205
- `X_LINK_NMB_OF_PROTOCOLS` (*depthai.XLinkProtocol* attribute), 205

[X\\_LINK\\_PCIE](#) (*depthai.XLinkProtocol attribute*), 205  
[X\\_LINK\\_UNBOOTED](#) (*depthai.XLinkDeviceState attribute*), 202  
[X\\_LINK\\_USB\\_CDC](#) (*depthai.XLinkProtocol attribute*), 205  
[X\\_LINK\\_USB\\_VSC](#) (*depthai.XLinkProtocol attribute*), 205  
[XLinkConnection](#) (*class in depthai*), 201  
[XLinkDeviceState](#) (*class in depthai*), 202  
[XLinkIn](#) (*class in depthai*), 202  
[XLinkOut](#) (*class in depthai*), 203  
[XLinkPlatform](#) (*class in depthai*), 204  
[XLinkProtocol](#) (*class in depthai*), 205  
[xmax\(\)](#) (*depthai.ImgDetection property*), 141  
[xmax\(\)](#) (*depthai.RawImageManipConfig.CropRect property*), 171  
[xmin\(\)](#) (*depthai.ImgDetection property*), 141  
[xmin\(\)](#) (*depthai.RawImageManipConfig.CropRect property*), 171

## Y

[y\(\)](#) (*depthai.Point2f property*), 163  
[y\(\)](#) (*depthai.Point3f property*), 163  
[y\(\)](#) (*depthai.Rect property*), 179  
[ymax\(\)](#) (*depthai.ImgDetection property*), 141  
[ymax\(\)](#) (*depthai.RawImageManipConfig.CropRect property*), 171  
[ymin\(\)](#) (*depthai.ImgDetection property*), 141  
[ymin\(\)](#) (*depthai.RawImageManipConfig.CropRect property*), 171  
[YoloDetectionNetwork](#) (*class in depthai*), 205  
[YoloSpatialDetectionNetwork](#) (*class in depthai*), 206  
[YUV400p](#) (*depthai.ImgFrame.Type attribute*), 145  
[YUV400p](#) (*depthai.RawImgFrame.Type attribute*), 176  
[YUV420p](#) (*depthai.ImgFrame.Type attribute*), 145  
[YUV420p](#) (*depthai.RawImgFrame.Type attribute*), 176  
[YUV422i](#) (*depthai.ImgFrame.Type attribute*), 145  
[YUV422i](#) (*depthai.RawImgFrame.Type attribute*), 176  
[YUV422p](#) (*depthai.ImgFrame.Type attribute*), 145  
[YUV422p](#) (*depthai.RawImgFrame.Type attribute*), 176  
[YUV444i](#) (*depthai.ImgFrame.Type attribute*), 145  
[YUV444i](#) (*depthai.RawImgFrame.Type attribute*), 176  
[YUV444p](#) (*depthai.ImgFrame.Type attribute*), 145  
[YUV444p](#) (*depthai.RawImgFrame.Type attribute*), 176

## Z

[z\(\)](#) (*depthai.Point3f property*), 164