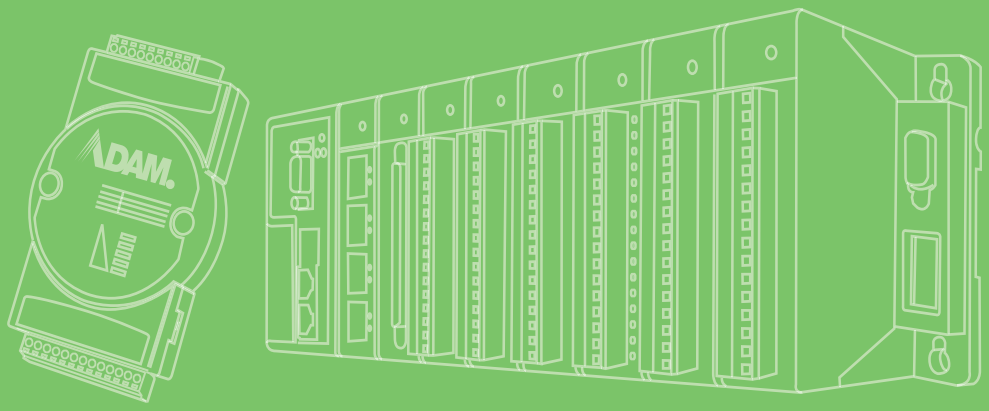


Programming Guide

ICAM-500 Series

Industrial AI Camera

ADVANTECH

Enabling an Intelligent Planet

Copyright

Acknowledgments

Intel and Pentium are trademarks of Intel Corporation.

Microsoft Windows and MS-DOS are registered trademarks of Microsoft Corp.

All other product names or trademarks are properties of their respective owners.

Product Warranty (2 years)

Advantech warrants to you, the original purchaser, that each of its products will be free from defects in materials and workmanship for two years from the date of purchase.

This warranty does not apply to any products which have been repaired or altered by persons other than repair personnel authorized by Advantech, or which have been subject to misuse, abuse, accident or improper installation. Advantech assumes no liability under the terms of this warranty as a consequence of such events.

Because of Advantech's high quality-control standards and rigorous testing, most of our customers never need to use our repair service. If an Advantech product is defective, it will be repaired or replaced at no charge during the warranty period. For out-of-warranty repairs, you will be billed according to the cost of replacement materials, service time and freight. Please consult your dealer for more details.

If you think you have a defective product, follow these steps:

1. Collect all the information about the problem encountered. (For example, CPU speed, Advantech products used, other hardware and software used, etc.) Note anything abnormal and list any onscreen messages you get when the problem occurs.
2. Call your dealer and describe the problem. Please have your manual, product, and any helpful information readily available.
3. If your product is diagnosed as defective, obtain an RMA (return merchandize authorization) number from your dealer. This allows us to process your return more quickly.
4. Carefully pack the defective product, a fully-completed Repair and Replacement Order Card and a photocopy proof of purchase date (such as your sales receipt) in a shippable container. A product returned without proof of the purchase date is not eligible for warranty service.
5. Write the RMA number visibly on the outside of the package and ship it prepaid to your dealer.

Part No. Edition 1

Printed in Taiwan June 2023

Declaration of Conformity

CE

This product has passed the CE test for environmental specifications when shielded cables are used for external wiring. We recommend the use of shielded cables. This kind of cable is available from Advantech. Please contact your local supplier for ordering information.

Test conditions for passing also include the equipment being operated within an industrial enclosure. In order to protect the product from damage caused by electrostatic discharge (ESD) and EMI leakage, we strongly recommend the use of CE-compliant industrial enclosure products.

FCC Class A

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference. In this event, users are required to correct the interference at their own expense.

Technical Support and Assistance

1. Visit the Advantech website at www.advantech.com/support to obtain the latest product information.
2. Contact your distributor, sales representative, or Advantech's customer service center for technical support if you need additional assistance. Please have the following information ready before you call:
   - Product name and serial number
   - Description of your peripheral attachments
   - Description of your software (operating system, version, application software, etc.)
   - A complete description of the problem
   - The exact wording of any error messages

Safety Precaution - Static Electricity

Follow these simple precautions to protect yourself from harm and the products from damage.

- To avoid electrical shock, always disconnect the power from your PC chassis before you work on it. Don't touch any components on the CPU card or other cards while the PC is on.
- Disconnect power before making any configuration changes. The sudden rush of power as you connect a jumper or install a card may damage sensitive electronic components.

Safety Instructions

1. Install the system only in area with restricted access.
2. Read these safety instructions carefully.
3. Retain this user manual for future reference.
4. Disconnect the equipment from all power outlets before cleaning. Use only a damp cloth for cleaning. Do not use liquid or spray detergents.
5. For pluggable equipment, the power outlet socket must be located near the equipment and easily accessible.
6. Protect the equipment from humidity.
7. Place the equipment on a reliable surface during installation. Dropping or letting the equipment fall may cause damage.
8. The openings on the enclosure are for air convection. Protect the equipment from overheating. Do not cover the openings.
9. Ensure that the voltage of the power source is correct before connecting the equipment to a power outlet.
10. Position the power cord away from high-traffic areas. Do not place anything over the power cord.
11. All cautions and warnings on the equipment should be noted.
12. If the equipment is not used for a long time, disconnect it from the power source to avoid damage from transient overvoltage.
13. Never pour any liquid into an opening. This may cause fire or electrical shock.
14. Never open the equipment. For safety reasons, the equipment should be opened only by qualified service personnel.
15. If any of the following occurs, have the equipment checked by service personnel:
    - The power cord or plug is damaged.
    - Liquid has penetrated the equipment.
    - The equipment has been exposed to moisture.
    - The equipment is malfunctioning, or does not operate according to the user manual.
    - The equipment has been dropped and damaged.
    - The equipment shows obvious signs of breakage.
16. Do not leave the equipment in an environment with a storage temperature of below -20 °C (-4 °F) or above 60 °C (140 °F) as this may damage the components. The equipment should be kept in a controlled environment.
17. CAUTION: Batteries are at risk of exploding if incorrectly replaced. Replace only with the same or equivalent type as recommended by the manufacturer. Discard used batteries according to the manufacturer's instructions.
18. In accordance with IEC 704-1:1982 specifications, the sound pressure level at the operator's position does not exceed 70 dB (A).

DISCLAIMER: These instructions are provided according to IEC 704-1 standards. Advantech disclaims all responsibility for the accuracy of any statements contained herein.

Contents

Chapter 1

Introduction

1.1 Introduction

The Advantech CAMNavi SDK offers the tools which provide developer/ ISV to operating ICAM-500 series Industrial AI camera and deploy the vision AI APP. based on NVIDIA Jetpack. The CAMNavi SDK uses Python language by default and is better adapted to image acquisition and AI algorithm integration.

1.2 Development environment

- The Advantech CAMNavi SDK and NVIDIA Jetpack goes with BSP of ICAM-500 series, user has to run CAMNavi SDK on the ICAM-500 series.
- Connect to HDMI display and USB 3 hub for keyboard mouse then ICAM-500 series is ready to development.
- To use CAMNavi SDK

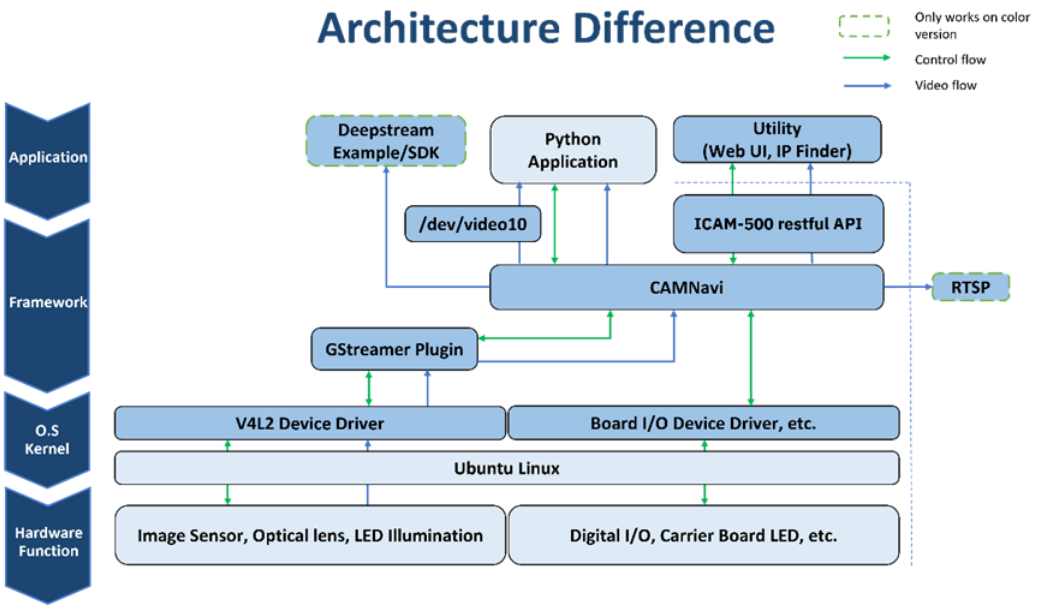
System requirement	ICAM-500		ICAM-520
OS	Linux Ubuntu 18.04		
Language	Python 3.6		
Jetpack version	4.4.1(Mono camera)	4.5.1(Color camera).	4.6.2

1.3 Limitations

- The CAMNavi SDK only operates on ICAM-500 series.
- To disable the web service before program ICAM-500 series with CAMNavi SDK since web service control video bus and I/O of ICAM-500 series for web utility. Follow the instruction of chapter 3 to disable the web service.

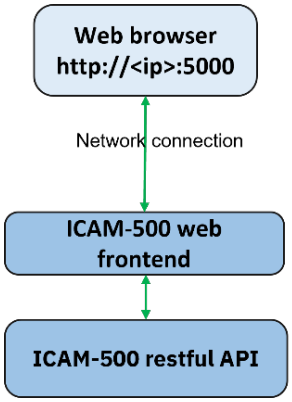
Chapter 2

ICAM-500 Series SDK



2.1 ICAM-500 Series Web Service

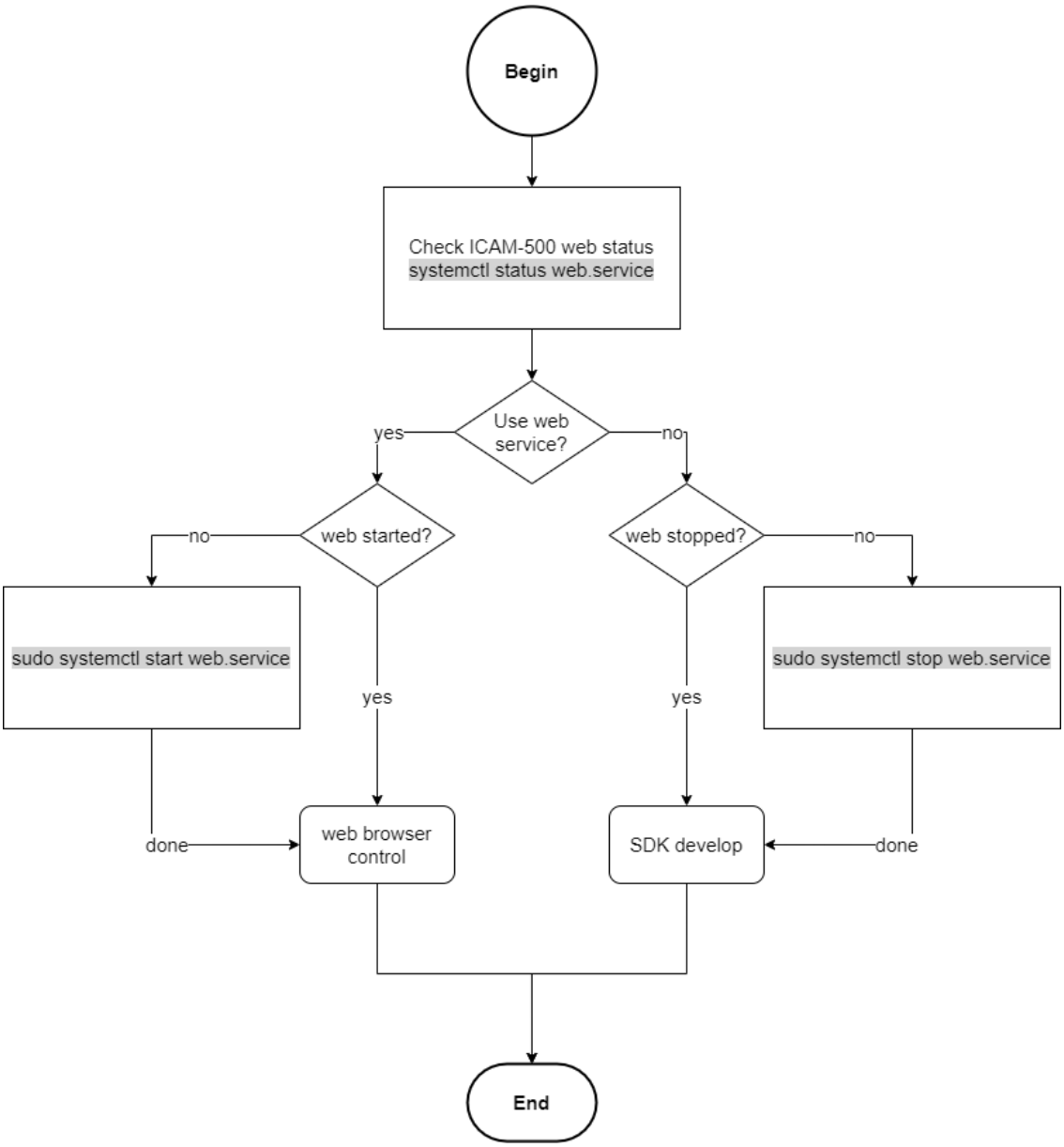
ICAM-500 series could use Web browser (Chrome) to operate from its web service. The service use **ICAM-500 series restful API** to control functions of ICAM-500 series device.



2.2 ICAM-500 Series Web Service and CamNavi2 SDK Operation

ICAM-500 series web service uses CamNavi2 SDK to control ICAM-500 series device. Only one instance could control ICAM-500 series functions. If developer use SDK to writing python applications, web service should turn off to release SDK control.

This flow chart shows start/stop web.service decision path.



Chapter 3

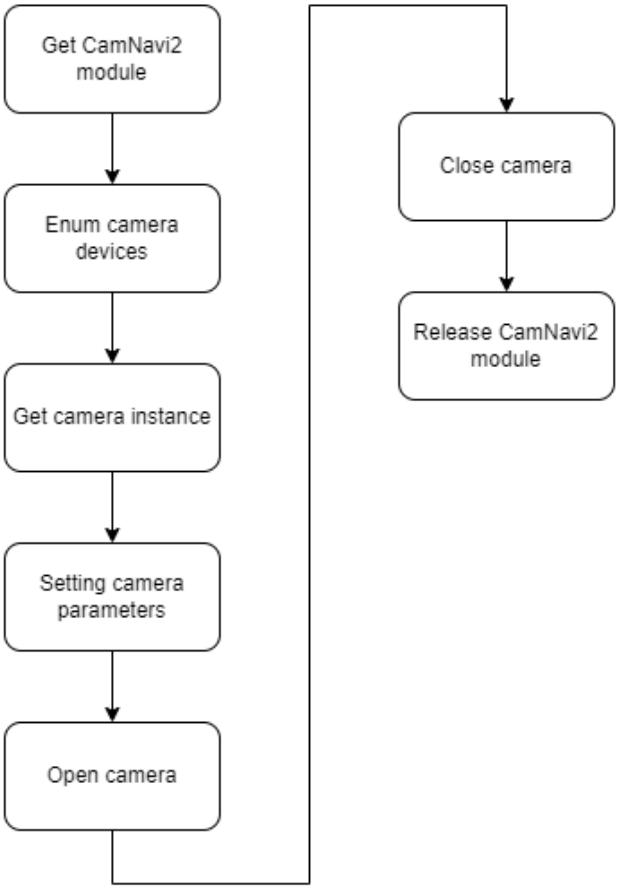
Operation Flow

Use ICAM-500 series python SDK should turn off web service.

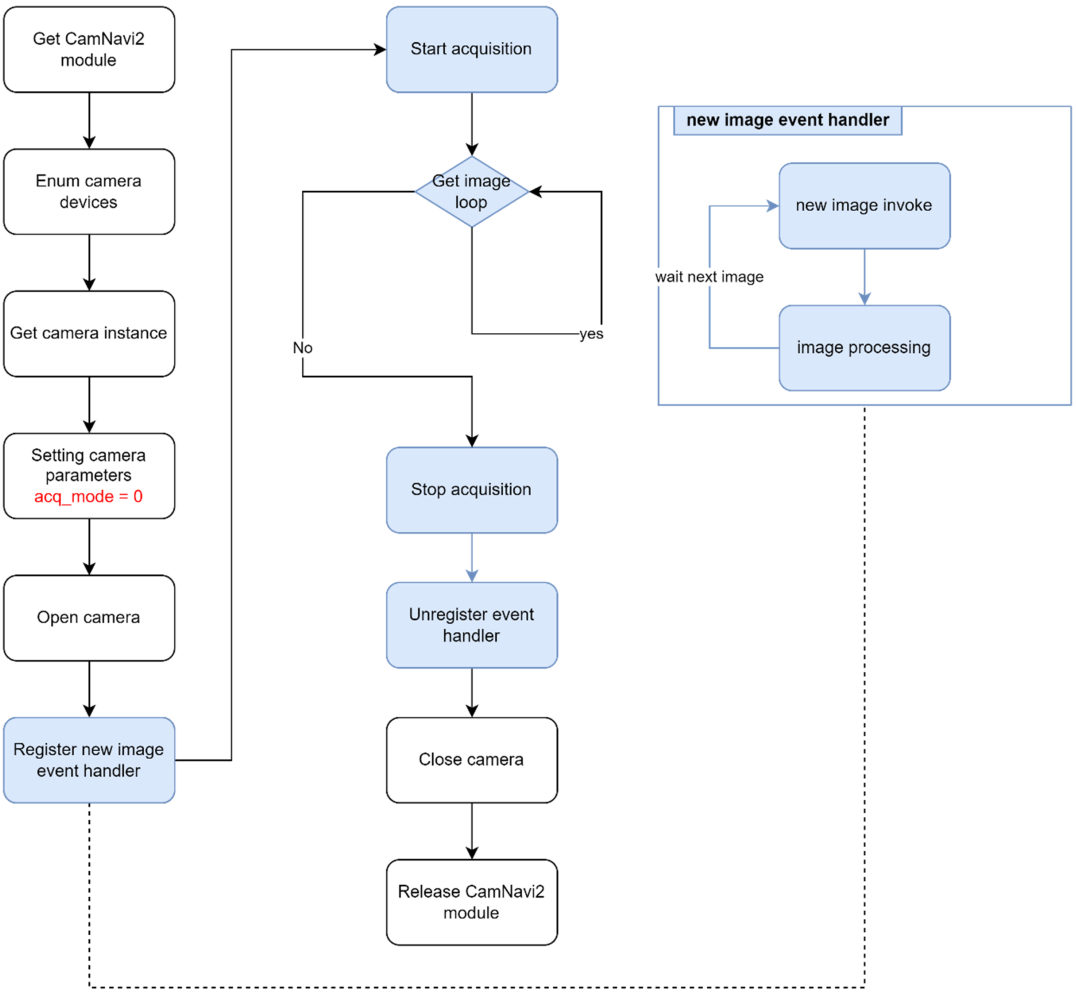
1. Temporary stop web service.
   $ sudo systemctl stop web.service

2. Stop iCAM-500 web service persistently.
   $ sudo systemctl stop web.service
   $ sudo systemctl disable web.service
   $ sudo systemctl disable autoui.service

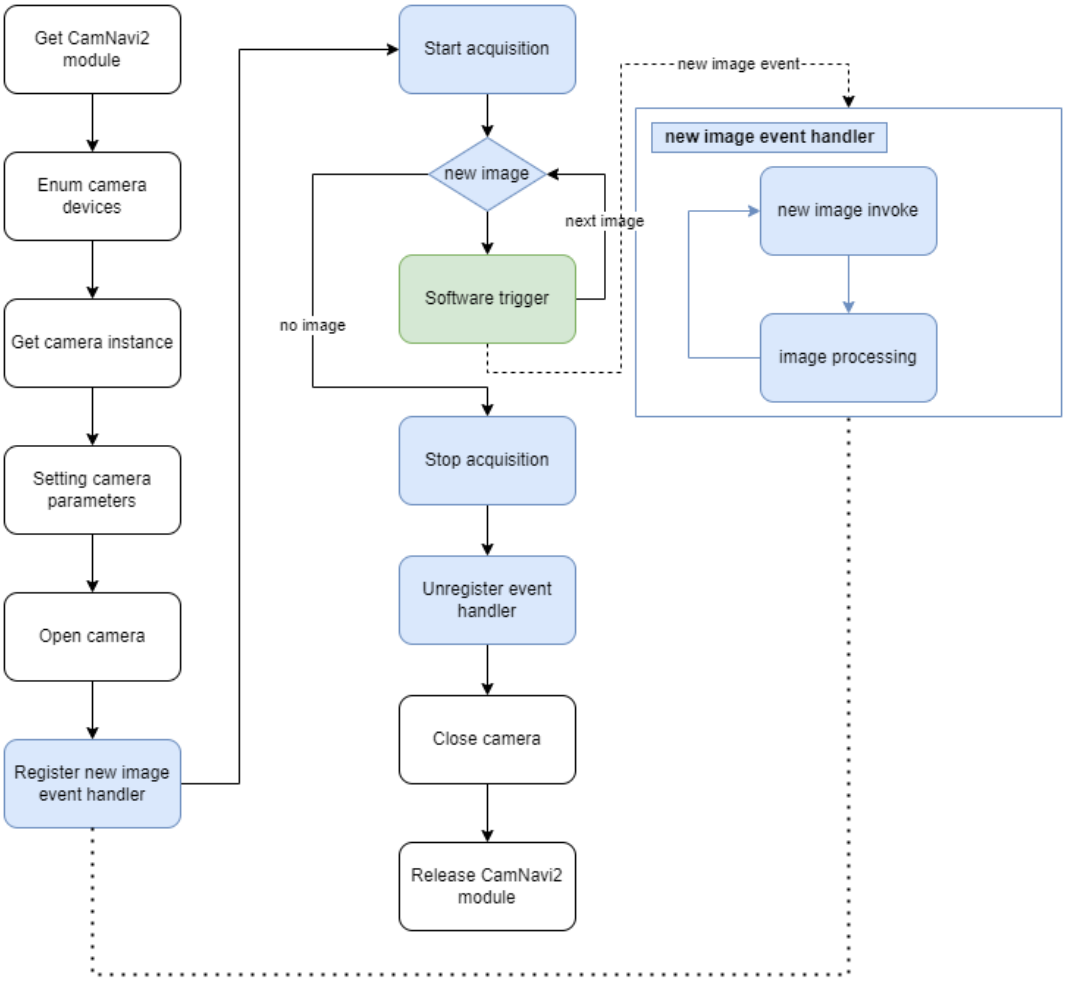
3.1 Open/Close Camera Flow



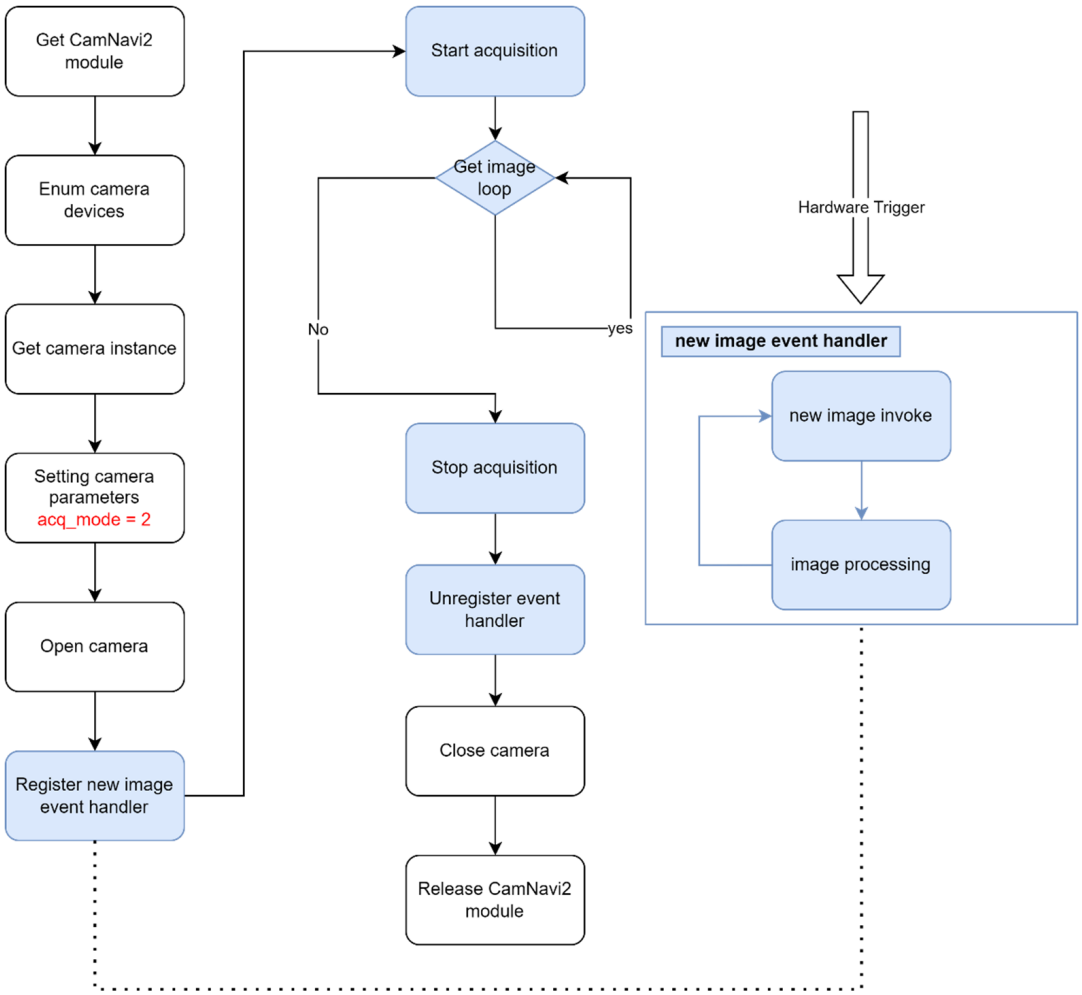
3.2 Image Acquisition



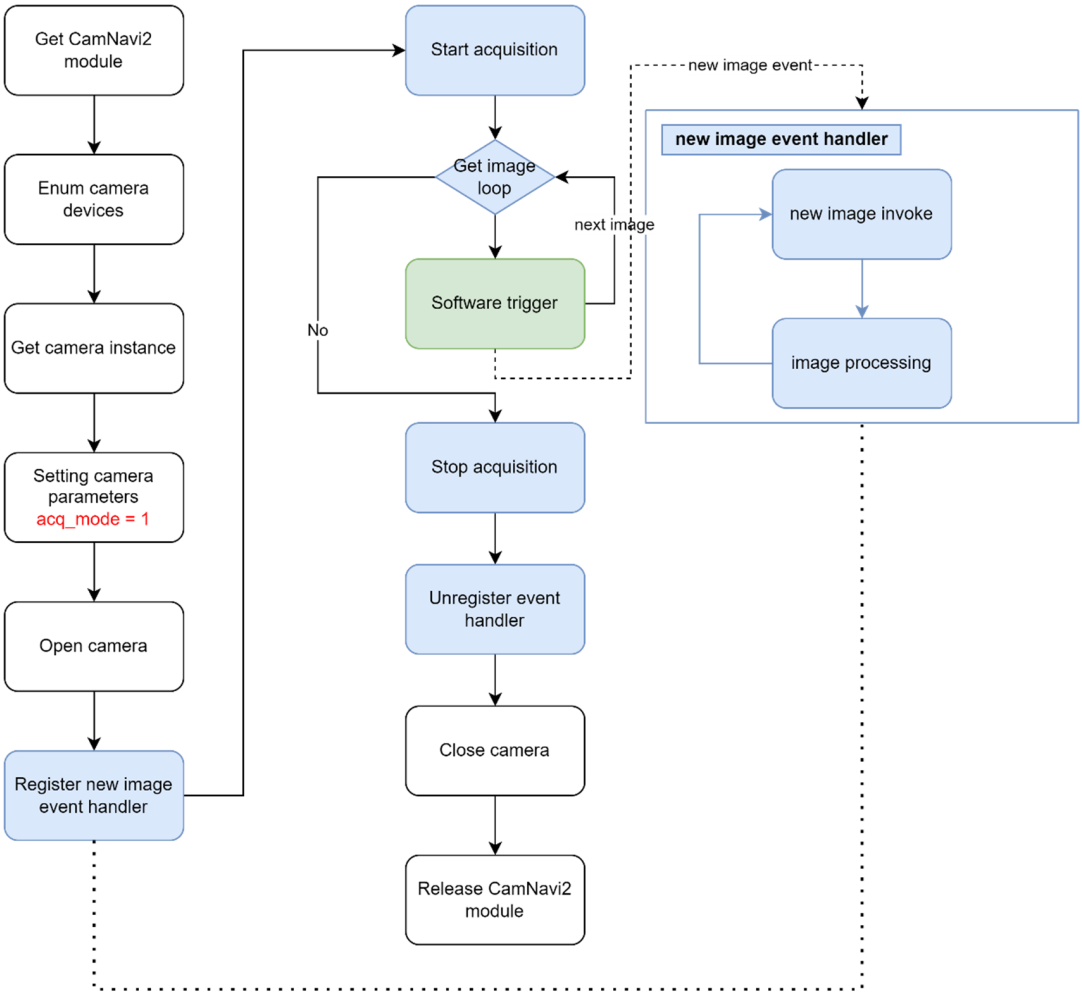
3.3 Image Trigger Flow



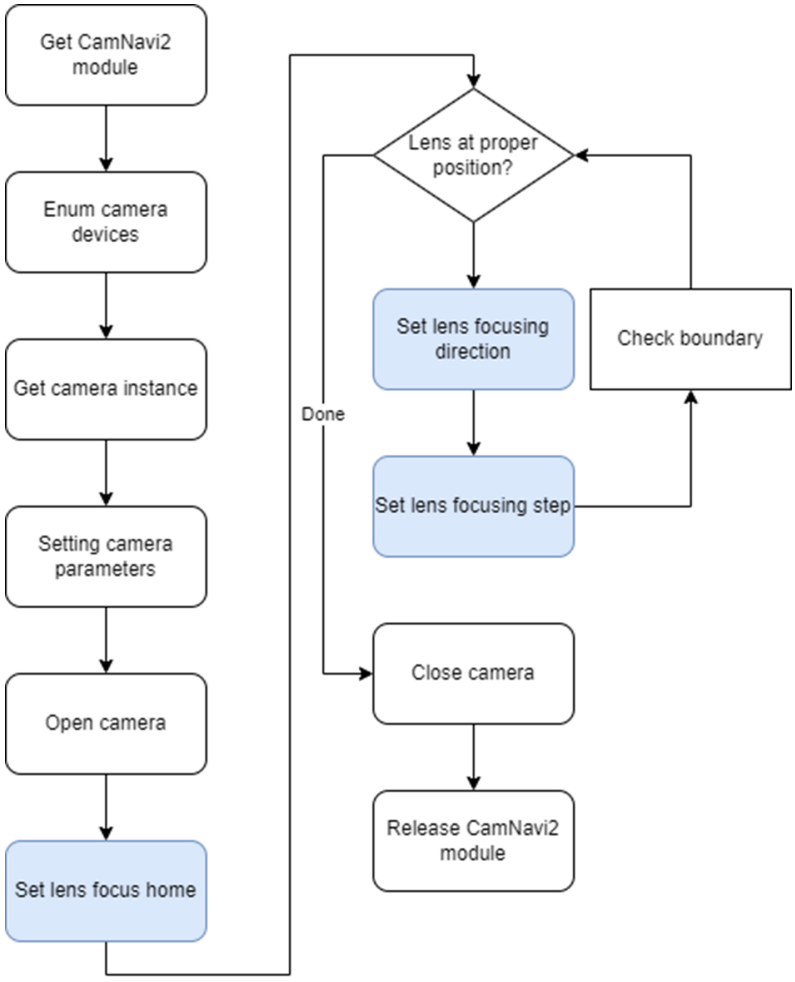
3.4 HW trigger Control flow



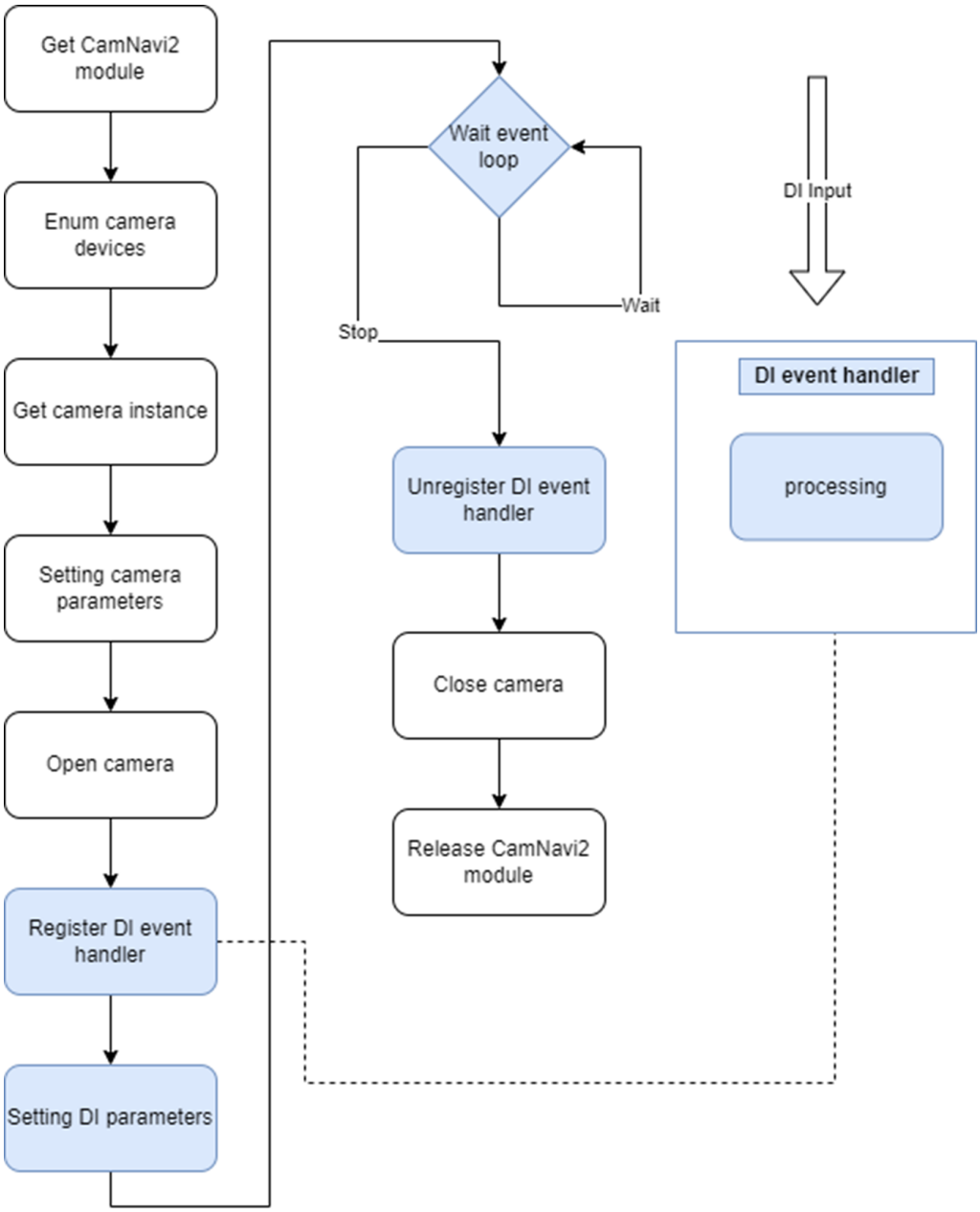
3.4.1 Software trigger



3.5 Lens Focusing Control



3.5.1 Digital Input flow

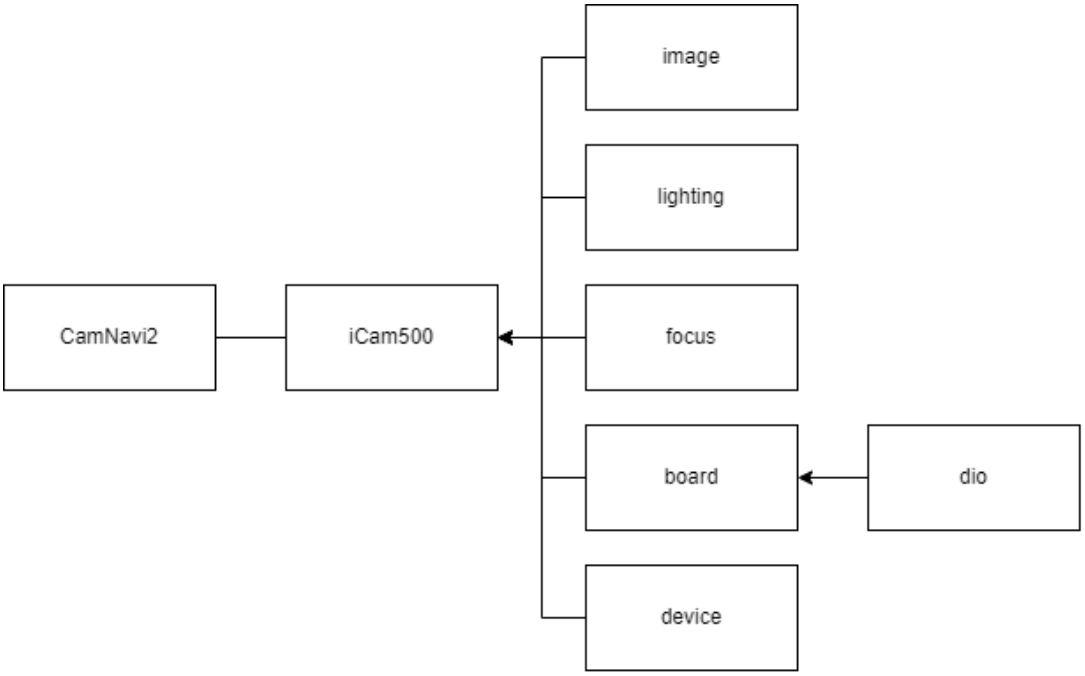


Chapter 4

CamNavi2 SDK API Manual

CamNavi2 SDK provide API of ICAM-500 series. Include sensor operation, lighting control, board control…

Below digram is SDK compoments hierarchy:



Module CamNavi2

Function	Description
get_device_by_name()	Use device name to get camera instance.
get_info()	Get BSP information.
get_fw_info()	Get firmware information.
advcam_config_pipeline()	Camera property configuration.
advcam_open()	Open camera instance.
advcam_close()	Close camera instance.
advcam_play()	Start image acquisition process. (Continuous or single shot)
advcam_stop()	Stop image acquisition process.
advcam_snap_image()	Snap an image from streaming. (10ms/image)
advcam_register_new_image_handler()	Register async new image event handler.

iCam500 Components

Module	Description
image	Configure frame properties.
focus	Lens focusing operation.
lighting	LED lighting control.
dev	ICAM-500 device configuration. (IP configuration)
board	Board IO control (digital input/output, buttons)

4.1 Requirement

4.1.1 Basic

Packages has contained in ICAM-500 BSP:

1. python 3
2. python3-gst-1.0
3. python3-networkmanager

4.1.2 Option

For ICAM-500 samples:

sudo apt install python3-opencv

4.2 module CamNavi2

4.2.1 class CamNavi2

class **CamNavi2.CamNavi2**(**args, **kwargs*)

CamNavi2 is a module for management ICAM device.

enum_camera_list()

Enumerate devices on iCam-500.

Returns: **dict**

A dict device name and device path mapping.

get_device_by_name(*name*)

Get device instance by device name.

Parameters: **name: str**

Device name

Returns: **ABSCamera**

Camera instance. If camera does not find, return None.

get_info()

General information provides by SDK. Provide BSP information.

Returns: **dict**

Information

get_fw_info()

Get camera module firmware version.

Returns: **str**

Firmware version string.

advcam_config_pipeline (cam, **pipe_params)

Camera gstreamer pipeline configuration.

Parameters: **params: dict{str, str}**

Set pipeline parameters.

'format': ['GRAY8', 'YUY2', 'BGRA']. Mono sku only has 'GRAY8'.
'acq_mode': 0 = Continuous, 1 = Software Trigger, 2 = Hardware Trigger
'width': 640
'height': 480
'frame_count': -1 (-1 = continuous, >0 = frame count)
'icam500=frame-rate': 60
'timestamp': [0, 1]. Print timestamp on image.
'pipeline_mode': ['default', 'simple']. default mode output jpeg image format, simple mode output jpeg image.

See also:

HYPERLINK \l "iCam500.iCam500.config_pipeline()

advcam_open(*cam, frame_count=-1*)

Open camera device.

Parameters: **cam: ABSCamera**

Camera instance from get_device_by_name.

frame_count: int

Burst images acquisition.

Returns: **str**

Device name.

None

Camera cannot open.

advcam_close(*cam*)

Close camera device.

Parameters: **cam: ABSCamera**

Camera instance

Returns: **int**

Close camera status.

- 0: success
- 1: camera instance is not existed.
- 2: camera get an exception.
- 3: camera is no opened.

advcam_get_capability(*cam*)

Camera capability information.

Returns: **dict**

Camera capability map

advcam_register_new_image_handler(*cam, new_image_cb*)

Register a new image event handler.
Event handler sample:

```
def new_sample_cb(sample):
    buf = smaple.get_buffer()
```

Parameters: **cam: ABSCamera**
Camera instance
new_sample_cb
New image event handler(callback).

Returns: **int**
0: Success.

advcam_play(*cam*)

Start image acquisition step. (Continuous mode or single shot)

Parameters: **cam: ABSCamera**
Camera instance

Returns: **bool**
True: success.

See also:
iCam500.iCam500.play()

advcam_snap_image(*cam, timeout=60*)

Snap an image in 10 ms period.

Parameters: **cam: ABSCamera**
Camera instance
timeout: int
Break waiting loop in timeout second.
unit: second

Returns: **None or [int]**
None: No image or timeout.
array: Image data in JPEG format

advcam_stop(*cam*)

Stop image acquisition procedure.

Returns: **bool**
False: pipeline does not set.
True: Success.

See also:
iCam500.iCam500.stop()

advcam_get_acq_mode(*cam*)

Get acquisition mode property.

Parameters: **cam: ABSCamera**

Camera instance

Returns: **int**

Acquisition mode:

- 0: Free run mode
- 1: Software trigger mode
- 2: Hardware trigger mode

advcam_set_acq_mode(*cam, acq_mode*)

Set acquisition mode property.

Parameters: **cam: ABSCamera**

Camera instance

acq_mode : int

Acquisition mode:

- 0: Free run mode
- 1: Software trigger mode
- 2: Hardware trigger mode

**** This API can be set at run-time, but need to re-open (ie. Call advcam_close(), then advcam_open()) to take effect*

advcam_get_focus_distance(*cam*)

Get lens focusing motor distance.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **int**

Lens focusing motor distance.

See also:

iCam500.iCam500.get_focus_distance()

advcam_set_focus_distance(*cam, distance*)

Set lens focusing motor distance.

Parameters: **cam: ABSCamera**

Camera instance.

distance: **int**

Lens focusing motor distance.
init value = 0
range = [0..1600]
default = 30

See also:

iCam500.iCam500.set_focus_distance()

advcam_get_lighting_pos(*cam*)

Get LED lighting location mode.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **int**

8 LED trigger location modes. range = [0..7]

See also:

iCam500.iCam500.get_lighting_pos()

advcam_set_lighting_pos(*cam, pos*)

Set LED lighting location mode.

Parameters: **cam: ABSCamera**

Camera instance.

pos: int

Trigger 8 LED lighting location.
init value = 3
range = [0..7]
0: off

Returns: **int**

0: Success

See also:

iCam500.iCam500.set_lighting_pos()

advcam_set_focus_distance(*cam, distance*)

Set LED lighting location mode.

Parameters: **cam: ABSCamera**

Camera instance.

distance: int

Lens focusing motor distance.
init value = 0
range = [0..300]
step = 30

See also:

HYPERLINK \l "iCam500.iCam500.set_focus_distance()

advcam_get_lighting_gain(*cam*)

Get LED lighting intensity.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **int**

LED lighting gain intensity.

See also:

iCam500.iCam500.get_lighting_gain()

advcam_set_lighting_gain(*cam, gain*)

Get LED lighting gain.

Parameters: **cam: ABSCamera**

Camera instance.

gain: int

LED lighting gain value.
init value = 9
range = [0..25]
step = 1

Returns: **int**

0: Success

See also:

iCam500.iCam500.set_lighting_gain()

advcam_get_lighting_strobe_enable(*cam*)

Get LED lighting enable state.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **int**

LED lighting enable state.
init value = 1
range [0..1]

See also:

HYPERLINK \l "iCam500.iCam500.get_lighting_strobe_enable()

advcam_set_lighting_strobe_enable(*cam, strobe*)

Set LED lighting enable state.

Parameters: **cam: ABSCamera**

Camera instance.
strobe:
enable = 1
disable = 0

Returns: **int**

0: Success

See also:

HYPERLINK \l "iCam500.iCam500.set_lighting_strobe_enable()

advcam_get_img_brightness(*cam*)

Get image brightness.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **int**

Sensor brightness value.

See also:

iCam500.iCam500.get_img_brightness()

advcam_set_img_brightness(*cam, brt*)

Set image brightness.

Parameters: **cam: ABSCamera**

Camera instance.

brt

Brightness value.
init value = 100
range = [0..255]
step = 1

Returns: **int**

0: Success

See also:

iCam500.iCam500.set_img_brightness()

advcam_get_img_gain(*cam*)

Get camera sensor gain value.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **int**

Camera sensor gain value.

advcam_set_img_gain(*cam, gain*)

Set camera sensor gain value.

Parameters: **cam: ABSCamera**

Camera instance.

gain

Camera sensor gain value.
init value = 0
range = [0..24]
step = 1

Returns: **int**

0: Success

advcam_get_img_sharpness(*cam*)

Get camera image sharpness value.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **int**

Camera image sharpness value.

advcam_set_img_sharpness(*cam, sharpness*)

Set camera image sharpness value. Support ‘GRAY8’ and ‘YUY2’ pixel format.

Parameters: **cam: ABSCamera**

Camera instance.

sharpness:

Camera image sharpness value.
init value = 0
range = [0..100]
step = 1

Returns: **int**

0: Success

advcam_status(*cam*)

Get camera instance status.

Parameters: **cam: ABSCamera**

Camera instance.

Returns: **CameraMode**

Reference iCam500.CameraMode

advcam_get_output_resolution(*cam*)

Get camera instance status.

Parameters: **cam: ABSCamera**
Camera instance.

Returns: **tuple(int, int)**
Get (width, height) tuple.

advcam_set_output_resolution(*cam, out_res*)

Set frame resolution.

Parameters: **cam: ABSCamera**
Camera instance.
out_res: tuple(int, int)
Tuple contains (width, height)

Returns: **tuple(int, int)**
Assigned value width, height
Raises: ValueError
width or height not in valid_resolution_list

**** This API can be set at run-time, but need to re-open (ie. Call advcam_close(), then advcam_open()) to take effect*

advcam_get_fw_info()

Get firmware information.

Returns: **str**
Firmware version information.

advcam_query_fw_sku(*cam*)

Get camera sku from firmware.

Parameters: **cam: ABSCamera**
Camera instance.

Returns: **str**
'0': Mono SKU FW
'1': Color SKU FW

advcam_reboot()

Reboot this camera device.

4.3 module iCam500

4.3.1 class iCam500

class **iCam500.iCam500**(**args, **kwargs*)

iCam500 camera class
Control items:

- camera sensor
- image streaming
- lens movement
- strobe
- digital I/O

valid_resolution_list

ICAM-500 supported resolution.
ICAM-500: {1408:1080, 1280:960, 640:480, 320:240, 1024:768} type:dict

image

Camera sensor properties control.
type: CImage

focus

Lens focusing motor control.
Lens movement properties.
type: Focus

lighting

Camera LED lighting control.
Strobe control on ICAM-500.
type: Lighting

dev

ICAM-500 device control.

- Device network configuration.
- Firmware update.

board

Device board I/O controls.

- Digital Input/Output. type: dio
- LED on board controls

type: Board

snap_image(*timeout=60 s*)

Snap an image.

Parameters: **timeout: int**

Wait image time.

Returns: **[]**

Image data in JPEG format.

software_trigger()

Invoke software trigger command.

property **hw_trigger_delay**

Hardware trigger delay time setting.

Sku: color

Getter: Read hardware trigger delay time value.

Setter: Set hardware trigger delay time value.

Value:
- init value: 0
- range: [0..65535]
- unit: us (microsecond)

Type: int

Raises: **ValueError**

Input value out of range.

property **hw_trigger_exp_src**

Set hardware trigger exposure source.

Sku: color

Getter: Read hardware trigger exposure source.

Setter: Set hardware trigger exposure source.

Value:
- init value: 0
- range: [0,1]
- 0: Exposure time from F/W
- 1: Exposure time from hardware trigger puls

Type: int

Raises: **ValueError**

Input value out of range.

get_pipeline_params(**_params_)

Get properties value in defined pipeline.

```
# Get width and height of image
params = {'width': 0, 'height': 0}
params = camera.get_pipeline_params(params)
for key in params:
    print("key %s, value %d" % (key, params[key]))
```

Parameters: **params: dict{str, str}**

Dict of pipeline paramters.

Returns: **dict{str, str}**

config_pipeline(**_params_)

Configure ICAM-500 gstreamer pipeline in defined format.

```
# Set resolution 640x480 in free run image
params = {'acq_mode':0, 'width':640, 'height':480}
camera.config_pipeline(params)
```

Parameters: **params: dict{str, str}**

Set pipeline parameters.
'format': ['YUY2', 'BGRA']. Mono sku only has 'GRAY8'.
'acq_mode': 0 = Continuous, 1 = Software Trigger, 2 = Hardware Trigger
'width': 640
'height': 480
'frame_count': -1 (-1 = continuous, >0 = frame count)
'icam500=frame-rate': 60
'timestamp': [0, 1]. Print timestamp on image.
'pipeline_mode': ['default', 'simple']. default mode output jpeg image format, simple mode output raw image.

Returns: **str**

"Config pipeline OK"

Notes

width and height parameter should in valid_resolution_list.
Ex. { 'width': 1408, 'height': 1080 }

get_device_name()

Get device name.

Returns: **str**

Device name.

open(*frame_count= -1*)

Open ICAM-500 device.

Parameters: **frame_count : int, optional**

- > 0: streaming stop when get frames up to frame_count.
- -1: free run streaming.

Returns: **dict{str, str}**

Device name.

close(*quiet=0*)

Close ICAM-500 device.

Parameters: **quiet: int**

- 0: default
- 1: Force turn off lighting.

Returns: **int**

- 0: Success.
- 1: Error in exception.

register_new_sample(*new_sample_cb*)

Register a new image event handler.
Event handler sample:

```
def new_sample_cb(sample):
    buf = smaple.get_buffer()
```

Parameters: **new_sample_cb**

New image event handler(callback).

Returns: **int**

0: Success.

play()

Start camera acquisition.

Returns: **bool**

True: Success

See also:

CamNavi2.CamNavi2.advcam_play()
Wrapped function in CamNavi2.

stop()

Stop image acquisition procedure.

Returns: **bool**

False: pipeline does not set
True: Success

See also:

CamNavi2.CamNavi2.advcam_stop()
Wrapped function in CamNavi2

s**et_acq_frame_rate**(*new_fps*)

Set acquisition frame rate.

Parameters: **new_fps: int**

Set frame rate.

Returns: **int**

Frame rate setting value.

get_lighting_pos()

Get LED lighting location mode.

Returns: **int**

8 LED trigger location modes.
range = [0..7]

See also:

lighting.Lighting.selector
Control by lighting instance.

set_lighting_pos(*pos)*

Set LED lighting location mode.

Parameters: **pos: int**

Trigger 8 LED lighting location.
init value = 3
range = [0..7]
0: off

Returns: **int**

0: Success

See also:

lighting.Lighting.selector
Control by lighting instance.

get_lighting_gain()

Get LED lighting gain.

Returns: **int**

LED lighting gain value.

See also:

lighting.Lighting.gain
Control by lighting instance.

set_lighting_gain(*gain)*

Set LED lighting gain.

Parameters: **gain: int**

LED lighting gain value.
init value = 9
range = [0..25]
step = 1

Returns: **int**

0: Success

See also:

lighting.Lighting.gain
Control by lighting instance.

get_lighting_strobe_enable()

Get LED lighting enable state.

Returns: **int**

LED lighting enable state.
init value = 1
range [0..1]

See also:

lighting.Lighting.strobe_enable
Control by lighting instance.

set_lighting_strobe_enable(*strobe)*

Set LED lighting enable state.

Parameters: **strobe**

- enable = 1
- disable = 0

Returns: **int**

0: Success

See also:

lighting.Lighting.strobe_enable
Control by lighting instance.

get_img_brightness()

Get camera sensor brightness.

Returns: **int**

Sensor brightness value.

See also:

image.CImage.brightness
Control by image instance.

set_img_brightness(*brightness)*

Set camera sensor brightness.

Parameters: **brightness**

Brightness value.
init value = 100
range = [0..255]
step = 1

Returns: **int**

0: Success

See also:

image.CImage.brightness
Control by image instance.

get_img_gain()

Get camera sensor gain value.

Returns: **int**

Camera sensor gain value.

See also:

image.CImage.gain
Control by image instance.

set_img_gain(*gain)*

Set camera sensor gain value.

Parameters: **gain**

Camera sensor gain value.
init value = 0
range = [0..24]
step = 1

Returns: **int**

0: Success

See also:

image.CImage.gain
Control by image instance.

get_img_sharpness()

Get camera image sharpness value.

Returns: **int**

Camera image sharpness value.

See also:

HYPERLINK \l "image.CImage.sharpness
Control by HYPERLINK \l "image instance.

set_img_sharpness(*sharpness)*

Set image sharpness value. Support ‘GRAY8’ and ‘YUY2’ pixel format.

Parameters: **gain**

Image sharpness value.
init value = 0
range = [0..100]
step = 1

Returns: **int**

0: Success

See also:

HYPERLINK \l "image.CImage.sharpness
Control by HYPERLINK \l "image instance.

get_img_exposure_time()

Get camera sensor exposure time value.

Returns: **int**

Camera sensor exposure time. Convert equation: value * 100 (micro sec.)

See also:

image.CImage.exposure_time
Control by image instance.

set_img_exposure_time(*exp_time)*

Set camera sensor exposure time value.

Parameters: **exp_time**

Camera sensor exposure value.
init value = 50
range = [1..1000]
step = 1 Convert equation: exp_time * 100 (micro sec.)

Returns: **int**

new exposure time

See also:

image.CImage.exposure_time
Control by image instance.

get_img_auto_exposure()

Get camera auto exposure fuction status.

Returns: **int**

Camera auto exposure function status.
- 1: Enabled
- 0: Disabled

See also:

image.CImage.auto_exposure
Control by image instance.

set_img_auto_exposure(*enable*)

Enable camera auto exposure function.

Parameters: **enable**

- 1: Enabled
- 0: Disabled

Returns: **bool**

Camera auto exposure function status.

- 1: Enabled
- 0: Disabled

See also:

image.CImage.auto_exposure
Control by image instance.

get_img_auto_exposure_range()

Get auto exposure range.

Returns: **tuple(int, int)**

Auto exposure range. (min, max) Value convert equation: value * 100 (micro sec.)

See also:

image.CImage.auto_exposure_range
Control by image instance.

set_img_auto_exposure_range(*min, max*)

Set auto exposure range.

Exposure time value spec:

init value = (50, 10000)
range = [1..10000]
step = 1 Convert equation: exp_time * 100 (micro sec.)

Parameters: **min**

Minimum value of auto exposure time.

max

Maximum value of auto exposure time.

Returns: **tuple(int, int)**

Auto exposure range. (min, max) Value convert equation: value * 100 (micro sec.)

See also:

image.CImage.auto_exposure_range
Control by image instance.

get_img_auto_gain()

Get camera auto gain function status.

Returns: **int**

Camera auto gain function status.

- 1: Enabled
- 0: Disabled

See also:

image.CImage.auto_gain
Control by image instance.

set_img_auto_gain(*enable*)

Enable camera auto gain function.

Parameters: **enable**

- 1: Enabled
- 0: Disabled

Returns: **bool**

Camera auto gain function status.

- 1: Enabled
- 0: Disabled

See also:

image.CImage.auto_gain
Control by image instance.

get_img_auto_gain_range()

Get auto gain range.

Returns: **tuple(int, int)**

Auto gain range. (min, max)

See also:

image.CImage.auto_gain_range
Control by image instance.

set_img_auto_gain_range(*min, max*)

Set auto gain range.

Gain value spec:

init value = (0, 24)
range = [0..24]
step = 1

Parameters: **min**

Minimum value of auto gain.

max

Maximum value of auto gain.

Returns: **tuple(int, int)**

Auto gain range. (min, max)

See also:

image.CImage.auto_gain_range
Control by image instance.

get_img_digital_gain()

Get camera digital gain value.

Returns: **int**
Camera digital gain value.

See also:

image.CImage.digital_gain
Control by image instance.

set_img_digital_gain(*new_dgain*)

Set camera digital gain value.

Parameters: **new_dgain**
New camera digital gain.

Returns: **int**
New digital gain value.

See also:

image.CImage.digital_gain
Control by image instance.

get_focus_distance()

Get lens focusing motor distance.

Returns: **int**
Lens focusing motor distance value.

See also:

focus.Focus.distance
Control by focus instance.

set_focus_distance(*distance*)

Set lens focusing motor distance.

Parameters: **distance: int**
Lens focusing motor distance.
init value = [30]
range = [0..300]

Returns: **int**
Assigned distance value.

See also:

focus.Focus.distance
Control by focus instance.

get_focus_direction()

Get lens focusing motor movement direction.

Returns: **int**

- 0: Zoom in
- 1: Zoom out

See also:

focus.Focus.direction
Control by focus instance.

set_focus_direction(*new_direction*)

Set lens focusing motor movement direction.
And invoke set_focus_distance(step) to move lens.

Parameters: **new_direction**

Change motor direction.

- 0: Zoom in
- 1: Zoom out

Returns: **int**

New direction value.

See also:

focus.Focus.direction
Control by focus instance.

get_led_color()

Get board LED color value.

Returns: **int**

- 0: off
- 1: green
- 2: orange
- 3: yellow

set_led_color(*new_color*)

Set board LED color.

Parameters: **new_color**

Change LED color

- 0: off
- 1: green
- 2: orange
- 3: yellow

Returns: **int**

New LED color value.

get_status()

Get camera status.
Camera connection and acquisition status.

Returns: **CameraMode**

Reference iCam500.CameraMode

get_output_ress()

Get current frame resolution.

Returns: **tuple(int, int)**

Get width and height tuple.

set_output_res(*out_res*)

Set frame resoultion.

Parameters: **out_res : tuple(int, int)**

Tuple contains (width, height)

Returns: **tuple(int, int)**

Assigned value width, height

Returns: **ValueError**

width or height not in valid_resolution_list

get_roi()

Get frame ROI value.

Returns: **tuple(int, int, int, int)**

Current image frame ROI value. tuple(top, left, right, bottom)

set_roi(*roi*)

Change frame image ROI value. Range value have to less than resolution value.

Parameters: **roi: tuple(int, int, int, int)**

New image ROI value. tuple(top, lef, right, bottom)

Returns: **bool**

True: Success.

set_timestamp_switch(*en=1*)

Enable timestamp on output image.

Parameters: **en: int**

Parameters:
0: Disable
1: Enable

Returns: **bool**

True: Success.

get_timestamp_switch()

Get timestamp on output image state.

Returns: **int**

0: Disable
1: Enabled

pos_zero()

Set lens focusing motor to home.

See also:

focus.Focus.pos_zero()
Control by focus instance.

focus_abs_position()

Get lens focusing motor abs position.

Returns: **int**

Motor position.

See also:

focus.Focus.position()
Control by focus instance.

Rfps()

Get SDK receiver FPS.

Returns: **float**

SDK receiver frame rate

4.3.2 enum CameraMode

class **iCam500.CameraMode**(*value*)

Camera device status

Disconnected = *0*

Camera disconnected

Connected = *1*

Camera connected

Playing = *2*

Camera start acquisition

Paused = *3*

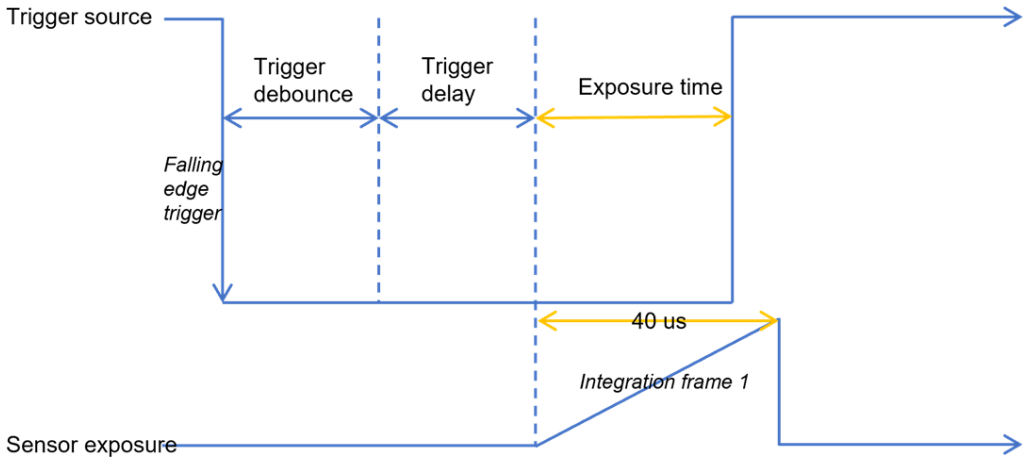
Camera stop acquisition

4.4 module acquisition

class **acquisition.Acquisition**(*camera*)

property **line0_debounce_time**

Hardware trigger debounce time setting.



Getter: Read debounce time value.

Setter: Write debounce time value.

Value:
- init value: 50
- range: [0..65535]
- step: 1
- unit: us (microsecond)

Type: int

Raises: **ValueError**

Input value out of range.

See also:

iCam500.iCam500.hw_trigger_delay
iCam500.iCam500.hw_trigger_exp_src

4.5 module image

4.5.1 class CImage

class **image.CImage**(*camera*)

Camera image properties controls.

property **brightness**

Image brightness setting.

Getter: Read sensor brightness setting.

Setter: Set sensor brightness setting.

Value:
- init value: 100
- range: [0..255]

Type: int

Raises: **ValueError**

Input value out of range.

property **gain**

Image gain setting.

Getter: Read sensor gain setting.

Setter: Set sensor gain setting.

Value:
- init value: 0
- range: [0..24]

Type: int

Raises: **ValueError**

Input value out of range.

property **saturation_op**

Color image enable saturation setting.

Sku: color

Getter: Read enable saturation setting.

Setter: Enable saturation setting.

Value:
- init value: 1
- range: [0,1]

Type: int

Raises: **ValueError**

Input value out of range.

property **saturation**

Color image saturation value. Enable property **color_trans_op**.

Sku: color

Getter: Read saturation value.

Setter: Set saturation value.

Value:
- init value: 128
- range: [0..255]

Type: int

Raises: **ValueError**
Input value out of range.

property **sharpness**

Image sharpness setting. Support 'GRAY8' and 'YUY2' pixel format.

Getter: Read sensor sharpness setting.

Setter: Set sensor sharpness setting.

Value:
- init value: 0
- range: [0..100]

Type: int

Raises: **ValueError**
Input value out of range.

property **gamma**

Image gamma setting.

Getter: Read sensor gamma setting.

Setter: Set sensor gamma setting.

Value:
- init value: 100
- range: [0..400]

Type: int

Raises: **ValueError**
Input value out of range.

property **hue**

Color image hue value. Enable property **color_trans_op**.

Sku: color

Getter: Read hue value.

Setter: Set hue value.

Value:
- init value: 128
- range: [0..255]

Type: int

Raises: **ValueError**
Input value out of range.

property **awb_op**

Color image enable auto white balance setting.

Sku: color

Getter: Read enable auto white balance setting.

Setter: Enable auto white balance setting.

Value:
- init value: 1
- range: [0,1]
- 0: Off, manual mode
- 1: Auto mode

Type: int

Raises: **ValueError**
Input value out of range.

property **awb_mode**

Color image auto white balance mode setting.

Sku: color

Getter: Read auto white balance mode setting.

Setter: Set auto white balance mode setting.

Value:
- init value: 0
- range: [0,1]
- 0: Narrow Mode
- 1: Wide Mode

Type: int

Raises: **ValueError**
Input value out of range.

property **awb_rgb**

Read color image auto white balance value.
Example: (r, g, b) = camera.image.awb_rgb

Sku: color

Getter: Read auto white balance value.

Type: tuple(red, green, blue)

Raises: **ValueError**
Input value out of range.

property **awb_red**

Set color image AWB red channel value.

Sku: color

Setter: Set AWB red channel value in image.awb_op manual mode.

Value:
- init value: 1024
- range: [1..16376]

Type: int

Raises: **ValueError**
Input value out of range.

property **awb_green**

Set color image AWB green channel value.

Sku: color

Setter: Set AWB green channel value in image.awb_op manual mode.

Value:
- init value: 1024
- range: [1..16376]

Type: int

Raises: **ValueError**

Input value out of range.

property **awb_blue**

Set color image AWB blue channel value.

Sku: color

Setter: Set AWB blue channel value in image.awb_op manual mode.

Value:
- init value: 1024
- range: [1..16376]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_op**

Color image enable color transform setting. CCM (color correction matrix). Gain00, Gain10 and Gain20 are group for R channel. Gain01, Gain11 and Gain21 are group for G channel. Gain02, Gain12 and Gain22 are group for B channel.

Sku: color

Getter: Read enable color transform setting.

Setter: Enable color transform setting.

Value:
- init value: 1
- range: [0,1]
- 0: Disable
- 1: Enable

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_00**

Color image color transform 00 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 00 value.

Setter: Set color transform 00 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_01**

Color image color transform 01 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 01 value.

Setter: Set color transform 01 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_02**

Color image color transform 02 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 02 value.

Setter: Set color transform 02 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_10**

Color image color transform 10 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 10 value.

Setter: Set color transform 10 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_11**

Color image color transform 11 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 11 value.

Setter: Set color transform 11 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_12**

Color image color transform 12 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 12 value.

Setter: Set color transform 12 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_20**

Color image color transform 20 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 20 value.

Setter: Set color transform 20 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**

Input value out of range.

property **color_trans_21**

Color image color transform 21 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 21 value.

Setter: Set color transform 21 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**
Input value out of range.

property **color_trans_22**

Color image color transform 22 value. CCM (color correction matrix)
[C00, C01, C02]
[C10, C11, C12]
[C20, C21, C22]

Sku: color

Getter: Read color transform 22 value.

Setter: Set color transform 22 value.

Value:
- init value: 1024
- range: [-4096..4096]

Type: int

Raises: **ValueError**
Input value out of range.

property **width**

Image width setting.

Getter: Read image width setting.

Setter: Set image width setting.

Value:
- init value: 640
- range: [320, 640, 1280, 1408]

Type: int

Raises: **ValueError**
Input value out of range.

property **height**

Image height setting.

Getter: Read image height setting.

Setter: Set image height setting.

Value:
- init value: 480
- range: [240, 480, 960, 1080]

Type: int

Raises: **ValueError**

Input value out of range.

property **exposure_time**

Camera sensor exposure time setting.

Getter: Read sensor exposure time setting.

Setter: Set sensor exposure time.

Value:
- init value: 50
- range: [10..10000]

Type: **int**

Raises: **ValueError**

Input value out of range.

property **auto_exposure_value**

Read sensor auto exposure value. (Read-only)

Getter: Read sensor auto exposure value.

Type: **int**

Raises: **ValueError**

Input value out of range.

property **auto_exposure**

Sensor auto exposure.

Getter: Read sensor auto exposure setting.

Setter: Enable sensor auto exposure setting.

Value:
- init value: 0
- range: [0, 1]

Type: **int**

Raises: **ValueError**

Input value out of range.

property **auto_exposure_range**

Camera sensor auto exposure time range setting.

Getter: Read sensor exposure time range setting.

Setter: Set sensor exposure time range.

Value:
- tuple(min, max)
- range: [1..10000]

Type: tuple(int, int)

Raises: **ValueError**
- Input value out of range.
- Minimum greater than maximum value.

property **auto_gain_value**

Read sensor auto gain value. (Read-only)

Getter: Read sensor auto gain value.

Type: **int**

Raises: **ValueError**

Input value out of range.

property **auto_gain**

Sensor auto gain.

Getter: Read sensor auto gain setting.

Setter: Enable sensor auto gain setting.

Value:
- init value: 0
- range: [0, 1]

Type: int

Raises: **ValueError**

Input value out of range.

property **auto_gain_range**

Camera sensor auto gain range setting.

Getter: Read sensor gain range setting.

Setter: Set sensor gain range.

Value:
- tuple(min, max)
- range: [0..24]

Type: tuple(int, int)

Raises: **ValueError**
- Input value out of range.
- Minimum greater than maximum value.

property **digital_gain**

Camera sensor digital gain setting.

Getter: Read sensor digital gain setting.

Setter: Set sensor digital gain.

Value:
- init value: 0
- range: [-512..511]

Type: int

Raises: **ValueError**

Input value out of range.

property **pixel_format**

Camera sensor pixel format.

Getter: Read sensor pixel format setting.

Setter: Set sensor pixel format.

Value:
- init value: 'GRAY8' in mono sku, 'YUY2' in color sku.
- range: ['GRAY8', 'YUY2', 'GBRA']

Type: str

Raises: **ValueError**

Input value out of range.

property **x_mirror**

Sensor image x mirror.

Getter: Read sensor pixel format setting.

Setter: Enable sensor image x mirror.

Value:
- 0: Disable (default)
- 1: Enable X flip

Type: int

Raises: **ValueError**

Input value out of range.

property **flip_op**

Software image flip operation.

Getter: Read image flip state.

Setter: Write image flip property.

Value:
- init value: 0
- range: [0..5]

Type: int

Define:
- 0: no-flip
- 1: rotate clockwise 90 degrees
- 2: rotate 180 degrees
- 3: rotate counter-clockwise 90 degrees
- 4: flip horizontally
- 5: flip vertically

Version: Added after SDK version 1.1.26

4.6 module focus

4.6.1 class Focus

class **focus.Focus**(*camera*)

Camera image properties controls.

property **distance**

Lens focusing motor distance control.

Example:

```
camera.focus.pos_zero() # Set motor to position 0
camera.focus.direction = 1 # Motor zoom in
camera.focus.distance = 30 # Foreward 30
camera.focus.distance = 30 # Foreward 30
camera.focus.position() # Get position value is 60
```

Getter: Read motor movement step.

Setter: Set motor movement step.

Value:
- init value: 0
- range: [0..300]

Type: int

Raises: **ValueError**

Input value out of range.

property **direction**

Lens focusing motor movement direction control.

Getter: Read motor movement direction.

Setter: Set motor movement direction.

Value:
- 0: Backward
- 1: Foreward

Type: int

Raises: **ValueError**

Input value out of range.

position()

Get lens focusing motor position.

Returns: **int**

Motor position.

rst_position

Restore lens focusing motor position.

Sku: color

Parameters: steps: int

Lens focusing motor goto this position from home.range: [0..1600]

Raises: **ValueError**

Input value out of range.

pos_zero()

Set lens focusing motor to position 0.

4.7 module lighting

4.7.1 class Lighting

class **lighting.Lighting**(*camera*)

property **selector**

LED lighting location mode control. 8 LED trigger location modes.

Getter: Read LED lighting flash location.

Setter: Set LED lighting flash location.

Value:
- init value = 3
- range = [0..7]
- 0: off

Type: int

Raises: **ValueError**

Input value out of range.

property **gain**

LED lighting gain control.

Getter: Read LED lighting gain value.

Setter: Set LED lighting gain.

Value:
- init value = 9
- range = [0..25]
- step = 1

Type: int

Raises: **ValueError**

Input value out of range.

property **strobe_enable**

Enable LED lighting function.

Getter: Read LED lighting enable status.

Setter: Set LED lighting enable.

Value:
- enable = 1
- disable = 0

Type: int

Raises: **ValueError**

Input value out of range.

4.8 module board

4.8.1 class Board

class **board.Board**(*camera*)

ICAM-500 board I/O control module

dio

Digital I/O control instance.
type: DigitalIO

4.9 module digitalio

4.9.1 digital I/O code slice

```
# Get camera instance.
camera = cn2.get_device_by_name('iCam500')
# Open camera device
cn2.advcam_open(camera)

# Setting do0 parameters
camera.dio.do0.op_mode = 0 # DO op mode: user output
camera.dio.do0.user_output = 0 # DO low, DI high
camera.dio.do0.reverse = 0

# Setting di0 parameters
camera.dio.di0.register_event(di_handler) # Register di0 signal handler
camera.dio.di0.mode = 1 # Enable DI mode
camera.dio.di0.source = 1 # DI in rising edge
camera.dio.di0.debounce_time = 100

camera.dio.do0.user_output = 1 # DO high, DI low
time.sleep(1)
camera.dio.do0.user_output = 0 # DO low, DI high

# Verify DI handler invoke times
assert Counter == 1

# Close camera
cn2.advcam_close(camera)
di event handler
# Digital input signal handler
def di_handler(*arg):
    global Counter
    Counter += 1
```

4.9.2 class DigitalIO

class **digitalio.DigitalIO**(*board*)

di0

digital input 0.
type: DigitalInput

di1

digital input 1.
type: DigitalInput

do0

digital output 0
type: DigitalOutput

do1

digital output 1
type: DigitalOutput

4.9.2.1 class DigitalInput

class **lighting.Lighting**(*board, dev_name*)

property **level**

Digital input level state.

Getter: Read input level state.
Value:
- 1: high
- 0: low

Type: int

property **mode**

Enable digital input mode.

Getter: Read mode.
Setter: Enable mode.
Value:
- 1: Enable DI.
- 0: Off.

Type: int
Raises: **ValueError**
Input value out of range.

property **source**

DI source configuration.

Getter: Read DI source.
Setter: Set DI source.

Value:
- 0: None
- 1: rising edge
- 2: falling edge
- 3: both

Type: int

Raises: **ValueError**

Input value out of range.

property **debounce_time**

DI trigger signal debounce time.

Getter: Get debounce time value

Setter: Set debounce time

Value:
- range: [0..65535]
- unit: Milli second.

Type: int

Raises: **ValueError**

Input value out of range.

property **debounce_mode**

Configure DI trigger signal debounce mode.

Getter: Read DI debounce mode.

Setter: Enable DI debounce mode.

Value:
- 0: Disable
- 1: Enable

Type: int

register_event(*event_handle*)

Register DI trigger event handler.

Example:

```
def di_event():
    pass
```

Parameters: **event_handle:**

Event handler function. Reference di_event example.

4.9.2.2 class DigitalOutput

class **digitalio.DigitalOutput**(*board, dev_name*)

property **user_output**

User set output level. This property works on op_mode is 0

Getter: Read output signal level.

Setter: Set output signal level.

Value:
- 0: low level
- 1: high level

Raises: **ValueError**

Input value out of range.

property **op_mode**

DO signal generate mode

Getter: Read DO op mode.

Setter: Set DO op mode.

Value:
- 0: User output DO
- 1: DI bypass(diabled)

Raises: **ValueError**

Input value out of range.

property **reverse**

DO signal reverse.

Getter: Read DO reverse state.

Setter: Set DO reverse state.

Value:
- 0: disable
- 1: enable reverse

Raises: **ValueError**

Input value out of range.

Notes

When setting reverse. op_mode and user_output have to set again.

property **delay_time**

DO signal delay output.

Getter: Read delay time.

Setter: SSet DO delay time.

Value:
- range: [0..65535]
- unit: Milli second.

Type: **int**

Raises: **ValueError**

Input value out of range.

4.10 module device

4.10.1 class Network

class **device.Network**

info()

Get ICAM-500 device network configuration

Returns: **tuple(str, str, str, str, str)**

Input value out of range.
1. mode : ‘dhcp’ or ‘static’
2. IP: IPv4 address
3. netmask: IPv4 netmask
4. gateway: IPv4 gateway address
5. MAC adress: ICAM-500 MAC address

config(*mode='dhcp', ip=None, netmask=None, gateway=None*)

Configure ICAM-500 network

info()

Get ICAM-500 device network configuration

Parameters: **mode: str**

- ‘dhcp’: Set network in DHCP mode. ip, netmask, gateway must set None.
- ‘static’: Set a static IPv4 address.

ip: str

IPv4 address. ex. ‘192.168.0.100’

netmask: str

IPv4 netmask. ex. ‘255.255.255.0’

gateway: str

IPv4 gateway. ex. ‘192.168.0.1’

Raises: **AttributeError**

If mode is “dhcp”, ip/netmask/gateway must set None.

4.10.2 class Firmware

class **device.Firmware**(*camera*)

info()

Read firmware version

Returns: **str**

Firmware version in date format(YYYYMMDD). ex. ‘20220216’

4.11 Exception

iCam500 properties value out of range would raise ValueError exceptions.

```
try:
    camera.lighting.seletor = 8
except ValueError:
    print("lighting selector out of range")
```

Chapter 5

CamNavi2 SDK Samples in Python

Recommendation install python3-opencv for image processing in samples.

5.1 Grab Frame Continuous

Grab image frame by callback function.

Register image grabber callback function:

`CamNavi2.advcam_register_new_image_handler(camera, new_image_handler)` or `camera.register_new_sample(new_image_handler)`

Callback function snippet:

`def new_image_handler(sample)`: is callback funtion. sample parameter is image data.

Image conversion snippet:

`def gst_to_opencv(sample)`: this function decode image to cv2:Mat format

```
#!/usr/bin/env python3

import cv2
import numpy
import time

from CamNavi2 import CamNavi2

image_arr = None
icam_color = 0

def gst_to_opencv(sample):
    # Extract gray jpeg image from sample
    buf = sample.get_buffer()

    buffer = buf.extract_dup(0, buf.get_size())
    arr = numpy.frombuffer(buffer, dtype=numpy.uint8)
    if icam_color == 1:
        im = cv2.imdecode(arr, cv2.IMREAD_COLOR)
    else:
        im = cv2.imdecode(arr, cv2.IMREAD_GRAYSCALE)

    return im


# Handle new image income
def new_image_handler(sample):
    global image_arr
```

```
    if sample is None:
        return

    # Convert sample
    arr = gst_to_opencv(sample)
    image_arr = arr


if __name__ == '__main__':
    try:
        cn2 = CamNavi2.CamNavi2()
    except:
        cn2 = CamNavi2()

    # Config camera resolution
    # iCAM-500 valid resolution list: 1280x960, 640x480, 320x240
    width = 640
    height = 480

    camera_dict = cn2.enum_camera_list()
    print("\nEnum. available camera list: ", camera_dict)

    sdk_info = cn2.get_info()
    print("\nGeneric SDK information: ", sdk_info)

    # Get camera instance.
    camera = cn2.get_device_by_name('iCam500')

    # Get camera sku to check determine image channel
    # 0 mono, 1 color
    icam_color = int(cn2.advcam_query_fw_sku(camera))
    print('icam-500 sku: %d' % (icam_color))

    # Set camera resolution and acquisition method block.
    # init pipeline for a camera module before open it
    pipe_params = {'acq_mode':0, 'width':width, 'height':
height, 'enable_infer':0}
    if icam_color == 1:
        pipe_params['format'] = 'YUY2'
    # pipe_params = {} # using default settings
    pipe_str = cn2.advcam_config_pipeline(camera,
**pipe_params)
```

```
    # Open camera device, and get continuous image
    #cn2.advcam_open(camera)
    # Open camera device, and declare 100 image frame count
    cn2.advcam_open(camera, 100)

    # Register new image handler
    cn2.advcam_register_new_image_handler(camera,
new_image_handler)

    # Start image streaming.
    cn2.advcam_play(camera)
    cn2.advcam_set_img_brightness(camera, 50)
    cn2.advcam_set_img_gain(camera, 10)

    while True:
        try:
            if image_arr is not None:
                # Display image if image_arr has data
                cv2.imshow("appsink", image_arr)
                lastKey = cv2.waitKey(10)
                if lastKey == 27:
                    cv2.destroyAllWindows()
                    break
            else:
                break

        except KeyboardInterrupt:
            break

    # unregister new image handler
    cn2.advcam_register_new_image_handler(camera, None)
    # camera.close()
    cn2.advcam_close(camera)
```

5.2 Grab Frame by Software Trigger

Use software trigger to trigger image acquisition.

Image grabbing procedure:

Same as Grab Frame Continuous describe.

Assign acq_mode:

acq_mode assign 1 for software trigger mode. If assign 2 is hardware trigger.

Trigger acquisition:

Use `camera.software_trigger()` to trigger image acquisition.

```
#!/usr/bin/env python3

import cv2
import numpy
import time

from CamNavi2 import CamNavi2

image_arr = None
icam_color = 0

'''
Image acquisition in software trigger method.
Press "Enter" key to invoke software trigger.
Press "Esc" key to quit this program
'''

def gst_to_opencv(sample):
    # Extract gray jpeg image from sample
    buf = sample.get_buffer()

    buffer = buf.extract_dup(0, buf.get_size())
    arr = numpy.frombuffer(buffer, dtype=numpy.uint8)
    if icam_color == 1:
        im = cv2.imdecode(arr, cv2.IMREAD_COLOR)
    else:
        im = cv2.imdecode(arr, cv2.IMREAD_GRAYSCALE)

    return im


# Handle new image income
def new_image_handler(sample):
    global image_arr
```

```
    if sample is None:
        return

    # Convert sample
    arr = gst_to_opencv(sample)
    image_arr = arr
    #cv2.imwrite('./sample.jpg', arr) # write image



if __name__ == '__main__':
    try:
        cn2 = CamNavi2.CamNavi2()
    except:
        cn2 = CamNavi2()

    # Config camera resolution
    # iCAM-500 valid resolution list: 1280x960, 640x480, 320x240
    width = 640
    height = 480

    camera_dict = cn2.enum_camera_list()
    print("\nEnum. available camera list: ", camera_dict)

    sdk_info = cn2.get_info()
    print("\nGeneric SDK information: ", sdk_info)

    # Get camera instance.
    camera = cn2.get_device_by_name('iCam500')

    # Get camera sku to check determine image channel
    # 0 mono, 1 color
    icam_color = int(cn2.advcam_query_fw_sku(camera))
    print('icam-500 sku: %d' % (icam_color))

    # Set camera resolution and acquisition method block.
    # init pipeline for a camera module before open it
    # acq_mode: 1 software trigger mode
    pipe_params = {'acq_mode':1, 'width':width, 'height':
height, 'enable_infer':0}
    if icam_color == 1:
        pipe_params['format'] = 'YUY2'
    # pipe_params = {} # using default settings
    pipe_str = cn2.advcam_config_pipeline(camera,
**pipe_params)
```

```
    # Open camera device, and get continuous image
    cn2.advcam_open(camera)
    # Open camera device, and declare 100 image frame count
    # cn2.advcam_open(camera, 10)

    # Register new image handler
    cn2.advcam_register_new_image_handler(camera,
new_image_handler)

    # Start image streaming.
    cn2.advcam_play(camera)
    cn2.advcam_set_img_brightness(camera, 50)
    cn2.advcam_set_img_gain(camera, 10)
    exp_time = camera.image.exposure_time * 100 # us
    exp_time /= 1000000 # sec.
    print('exposure time: ', exp_time , ' sec')

    # Fetch first image
    camera.software_trigger()
    if exp_time < 0.5:
        time.sleep(0.5)
    while True:
        try:
            if image_arr is not None:
                # Display image if image_arr has data
                cv2.imshow("appsink", image_arr)
                lastKey = cv2.waitKey(10)
                if lastKey == 27: # Esc
                    cv2.destroyAllWindows()
                    break
                elif lastKey == 10: # Enter
                    camera.software_trigger()
            else:
                break

        except KeyboardInterrupt:
            break

    # unregister new image handler
    cn2.advcam_register_new_image_handler(camera, None)
    # camera.close()
    cn2.advcam_close(camera)
```

5.3 Digital Input Processing

Set DI mode and source:

`camera.dio.di0.mode` assign 1 to enable DI interrupt.

`camera.dio.di0.source` select DI invoke signal.

Register DI callback function:

`camera.dio.di0.register_event(di_handler)`

Callback function snippet:

`def di_handler(*arg):`

```
#!/usr/bin/env python3

import time

from CamNavi2 import CamNavi2

"""
Pin connection:

   do0 -------- di0

"""

Counter = 0
# Digital input signal handler
def di_handler(*arg):
    global Counter
    Counter += 1

if __name__ == '__main__':
    try:
        cn2 = CamNavi2.CamNavi2()
    except:
        cn2 = CamNavi2()

    # Config camera resolution
    # iCAM-500 valid resolution list: 1280x960, 640x480, 320x240
    width = 640
    height = 480

    camera_dict = cn2.enum_camera_list()
    print("\nEnum. available camera list: ", camera_dict)
```

```
    sdk_info = cn2.get_info()
    print("\nGeneric SDK information: ", sdk_info)

    # Get camera instance.
    camera = cn2.get_device_by_name('iCam500')

    # Set camera resolution and acquisition method block.
    # init pipeline for a camera module before open it
    # acq_mode: 1 software trigger mode
    pipe_params = {'acq_mode':1, 'width':width, 'height':
height, 'enable_infer':0}
    # pipe_params = {} # using default settings
    pipe_str = cn2.advcam_config_pipeline(camera,
**pipe_params)

    # Open camera device
    cn2.advcam_open(camera)

    # Setting do0 parameters
    camera.dio.do0.op_mode = 0 # DO op mode: user output
    camera.dio.do0.user_output = 0 # DO low, DI high
    camera.dio.do0.reverse = 0

    # Setting di0 parameters
    camera.dio.di0.invert = 0 # Disable invert
     camera.dio.di0.register_event(di_handler) # Register di0 signal
handler
    camera.dio.di0.mode = 1 # DI mode
    camera.dio.di0.source = 1 # DI invoke in rising edge
    camera.dio.di0.debounce_time = 100

    time.sleep(1)
    times = 100

    # DO simulate pulse to produce DI
    for i in range(0, times):
        camera.dio.do0.user_output = 1 # DO high, DI low
        time.sleep(1)
        camera.dio.do0.user_output = 0 # DO low, DI high
        time.sleep(1)

    # Verify DI handler invoke times
    assert Counter == times

    # Close camera
    cn2.advcam_close(camera)
```

5.4 Lens Focusing Control

Press “Enter” to grab image.
Press “+” to move lens forward.
Press “-” to move lens bakeward.
Press “Esc” to quit program.

```
#!/usr/bin/env python3

import cv2
import numpy
import time

from CamNavi2 import CamNavi2

image_arr = None
icam_color = 0

def gst_to_opencv(sample):
    # Extract gray jpeg image from sample
    buf = sample.get_buffer()

    buffer = buf.extract_dup(0, buf.get_size())
    arr = numpy.frombuffer(buffer, dtype=numpy.uint8)
    if icam_color == 1:
        im = cv2.imdecode(arr, cv2.IMREAD_COLOR)
    else:
        im = cv2.imdecode(arr, cv2.IMREAD_GRAYSCALE)

    return im

# Handle new image income
def new_image_handler(sample):
    global image_arr

    if sample is None:
        return

    # Convert sample
    arr = gst_to_opencv(sample)
    image_arr = arr
```

```
if __name__ == '__main__':
    try:
        cn2 = CamNavi2.CamNavi2()
    except:
        cn2 = CamNavi2()

    # Config camera resolution
    # iCAM-500 valid resolution list: 1280x960, 640x480, 320x240
    width = 640
    height = 480

    camera_dict = cn2.enum_camera_list()
    print("\nEnum. available camera list: ", camera_dict)

    sdk_info = cn2.get_info()
    print("\nGeneric SDK information: ", sdk_info)

    # Get camera instance.
    camera = cn2.get_device_by_name('iCam500')

    # Get camera sku to check determine image channel
    # 0 mono, 1 color
    icam_color = int(cn2.advcam_query_fw_sku(camera))
    print('icam-500 sku: %d' % (icam_color))

    # Set camera resolution and acquisition method block.
    # init pipeline for a camera module before open it
    # acq_mode: 1 software trigger mode
            pipe_params   =   {'acq_mode':1,   'width':width,
'height':height, 'enable_infer':0}
    if icam_color == 1:
        pipe_params['format'] = 'YUY2'
    # pipe_params = {} # using default settings
    pipe_str = cn2.advcam_config_pipeline(camera,
**pipe_params)

    # Open camera device, and get continuous image
    # cn2.advcam_open(camera)
    # Open camera device, and declare 100 image frame count
    cn2.advcam_open(camera, 10)

    # Register new image handler
    cn2.advcam_register_new_image_handler(camera,
new_image_handler)
```

```
    # Start image streaming.
    cn2.advcam_play(camera)
    cn2.advcam_set_img_brightness(camera, 50)
    cn2.advcam_set_img_gain(camera, 10)

    # Set lens focusing motor to 0
    camera.focus.pos_zero()

    # Get first image
    camera.software_trigger()
    while True:
        try:
            if image_arr is not None:
                # Display image if image_arr has data
                cv2.imshow("appsink", image_arr)
                lastKey = cv2.waitKey(10)
                if lastKey == 27: # Esc
                    cv2.destroyAllWindows()
                    break
                elif lastKey == 10: # Enter
                    camera.software_trigger()
                elif lastKey == 43: # '+'
                    camera.focus.direction = 0 # lens focusing
motor forward
                    try:
                        camera.focus.distance = 30
                        print("lens motor posistion: ",
camera.focus.position())
                    except ValueError:
                        print("lens position out of index")
                    camera.software_trigger()
                elif lastKey == 45: # '-'
                    camera.focus.direction = 1 # lens focusing
motor backward
                    try:
                        camera.focus.distance = 30
                        print("lens motor posistion: ",
camera.focus.position())
                    except ValueError:
                        print("lens position out of index")

                    camera.software_trigger()
            else:
                break
```

```
        except KeyboardInterrupt:
            break

    # unregister new image handler
    cn2.advcam_register_new_image_handler(camera, None)
    # camera.close()
    cn2.advcam_close(camera)
```

5.5 Lighting Control

Control 8 LED lighting positions.

```
#!/usr/bin/env python3

import time

from CamNavi2 import CamNavi2

if __name__ == '__main__':
    try:
        cn2 = CamNavi2.CamNavi2()
    except:
        cn2 = CamNavi2()

    # Config camera resolution
    # iCAM-500 valid resolution list: 1280x960, 640x480, 320x240
    width = 640
    height = 480

    camera_dict = cn2.enum_camera_list()
    print("\nEnum. available camera list: ", camera_dict)

    sdk_info = cn2.get_info()
    print("\nGeneric SDK information: ", sdk_info)

    # Get camera instance.
    camera = cn2.get_device_by_name('iCam500')

    # Set camera resolution and acquisition method block.
    # init pipeline for a camera module before open it
    # acq_mode: 1 software trigger mode
            pipe_params   =   {'acq_mode':1,   'width':width,
'height':height, 'enable_infer':0}
    # pipe_params = {} # using default settings
    pipe_str = cn2.advcam_config_pipeline(camera,
**pipe_params)

    # Open camera device
    cn2.advcam_open(camera)
```

```
camera.acquisition.strobe0_op = 0
# camera.lighting.strobe_enable = 0
camera.lighting.gain = 50
# Changing lighting 8 position mode
for pos in range(0, 8):
    camera.lighting.selector = pos
    time.sleep(1)

# Close lighting
camera.lighting.selector = 0

# Close camera
cn2.advcam_close(camera)
```

5.6 Get Image from Video10

Use opencv to access video10 video stream.

```
#!/usr/bin/env python3
#
# Before running this sample
# Please ensure
# 1. you have the necessary package installed.
# ins_py36_dep.sh
# 2. icam-500 is at playing status
#
# when icam-500 is at playing at >= 5fps, there is another
rtsp stream available at port 8550
# rtsp://<ip address>:8550/video
#
# version:
#  20230116: Added save png and save raw image methods.


import cv2
import time

def show_cam():
    # icam-500 is playing and output frames to /dev/video10
    cam = cv2.VideoCapture(10)
    cam.set(cv2.CAP_PROP_FORMAT, -1)
    while True and cam.isOpened():
        ret, img = cam.read()
        if ret:
            cv2.imshow('icam-500', img)
            key = cv2.waitKey(1)
            if key == 27: # esc key
                break
            elif key == ord('s'): # press 's' to save image
                cv2.imwrite('./video10.png', img)

        else:
            print('failed to capture an image')
            time.sleep(0.2)

    cv2.destroyAllWindows()
    print('end program')

if __name__ == '__main__':
    show_cam()
```

ADVANTECH

Enabling an Intelligent Planet

www.advantech.com

Please verify specifications before quoting. This guide is intended for reference purposes only.