WRP Client

Feb 16, 2020

Classes:

Welcome to the WRP Client's documentation! WRP Client and WRP Server are two parts of a driver that allows to connect to the Workswell InfraRed Camera using Python. This repository contains the client part, that is written in Python. The second part, WRP Server, is written in C# because the Workswell company provides and supports access to the cameras only through their C# SDK and not throught any other language.

CHAPTER 1

Client

The class Client is implemented in file client.py and has this methods:

class wrpclient.client.**Client**

Represents client that can establish connection with the WRP server and ask for the list of available cameras

connect(*ip_address*, *port=8754*, *timeout=10*)

Connect client to the server with given IP address and port. ValueError exception is raised when client is already connected.

Params

- ip_address: str, IP address of the WRP server
- port: int, port of the WRP server
- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

connect_async(*ip_address*, *port*)

Asynchronously connect client to the server with given IP address and port. ValueError exception is raised when client is already connected.

Params

- ip_address: str, IP address of the WRP server
- port: int, port of the WRP server

Return

None

disconnect(*timeout=10*)

Disconnect client from the server. ValueError exception is raised when client is not connected.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

disconnect_async()
Asynchronously disconnect client from the server. ValueError exception is raised when client is not connected.

Params

None

Return

None

get_camera(*serial_number*, *timeout=10*)
Get camera with the given serial number. First client must be connected to the server. If he is not in the connected state, ValueError exception is raised. If there is no camera with the given serial number available, ValueError exception is also raised.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

instance of `Camera`

get_camera_async(*serial_number*)
Get camera with the given serial number. First client must be connected to the server. If he is not in the connected state, ValueError exception is raised. If there is no camera with the given serial number available, ValueError exception is also raised.

Params

None

Return

instance of `Camera`

get_cameras(*timeout=10*)
Get list of all cameras connected to the server. First client must be connected to the server. If he is not in the connected state, ValueError exception is raised.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

list of instances of `Camera`

get_cameras_async()
Asynchronously get list of all cameras connected to the server. First client must be connected to the server. If he is not in the connected state, ValueError exception is raised.

Params

None

Return

list of instances of `Camera`

is_connected()

Determine whether client is connected to the WRP server.

Params

None

Return

bool

CHAPTER 2

Camera

The class Camera is implemented in file camera.py and has this methods:

class wrpclient.camera.**Camera**(*connector*)

Represents camera with corresponding hardware on the server-side of application. Can be used to get frame or start continuous shot. Instances of this class should not be created by the user directly (camera = Camera()) but they should be obtained using client.get_cameras().

close(*timeout=10*)

Disconnect from the camera throught the server. ValueError exception is raised when camera is not open.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

close_async()

Asynchronously disconnect from the camera throught the server. ValueError exception is raised when camera is not open.

Params

None

Return

None

get_frame(*timeout=10*)

Get a single frame from the connected camera. ValueError exception is raised when camera is not connected first.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

2-dimensional numpy matrix with dtype float32 containing temperature values in °C

get_frame_async()
Asynchronously get a single frame from the connected camera. ValueError exception is raised when camera is not connected first.

Params

None

Return

2-dimensional numpy matrix with dtype float32 containing temperature values in °C

is_open()
Determine whether the camera is open by the WRP server for this client.

Params

None

Return

bool

open(*timeout=10*)
Connect to the camera throught the server. ValueError exception is raised when another camera is already open.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

open_async()
Asynchronously connect to the camera throught the server. ValueError exception is raised when another camera is already open.

Params

None

Return

None

start_continuous_shot(*callback*, *timeout=10*)
Start continuous grabbing from the camera. Each frame obtained from the camera will be asynchronously passed as argument to the given callback function. ValueError exception is raised when camera is not connected first.

Params

- callback: callable, function that will be repeatedly called in its own thread for each received frame
- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

start_continuous_shot_async(*callback*)

Asynchronously start continuous grabbing from the camera. Each frame obtained from the camera will be asynchronously passed as argument to the given callback function. ValueError exception is raised when camera is not connected first.

Params

- callback: callable, function that will be repeatedly called in its own thread for each received frame

Return

None

stop_continuous_shot(*timeout=10*)

Stop continuous grabbing from the camera. ValueError exception is raised when camera is not connected first or it is not in continuous grabbing state.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

stop_continuous_shot_async()

Asynchronously stop continuous grabbing from the camera. ValueError exception is raised when camera is not connected first or it is not in continuous grabbing state.

Params

None

Return

None

CHAPTER 3

Message

The class Message is implemented in file message.py and has this methods:

class wrpclient.message.**Message**

Structure that implements WRP message and is used for passing all the information at once

static convert_int_to_message_type(*message_type_value*)

Static method that converts a given integer to the Message.Type enum ValueError exception when a given integer is not associated with any Message.Type.

Params

- message_type_value: int

Return

Message.Type enum

static create_message(*message_type*, ***kwargs*)

Static method that creates new message and sets its attributes by given values. Also add this values into bytes[] with correct order. ValueError exception is raised message type and other attributes does not match according to WRP.

Params

- message_type: Message.Type
- kwargs: dict that should contain values depending on the type of the message

Return

instance of *Message*

static create_message_from_buffer(*message_type_value*, *payload=b''*, *payload_length=0*)

Static method that checks if the given message type and payload extracted from a socket correct and decompose it to attributes specific for each message type ValueError exception is raised when given payload's length and payload_length does not match according to WRP.

Params

- message_type_value: int, code of the message type
- payload: bytes, extracted from the socket
- payload_length: int, length of the payload according to received bytes from the socket

Return

instance of *Message*

encode()
Encode message along its attributes (message type, payload etc.) to bytearray

Params

None

Return

bytearray containing encoded message

static is_int_valid_message_type(*message_type_value*)
Static method that checks if a given integer represents valid message type

Params

- message_type_value: int

Return

bool

static xml_to_camera_list(*connector*, *xml*)
Static method that convert XML received in CAMERA_LIST message to the list of instances of cameras. ValueError exception is raised when XML is not valid according to WRP.

Params

- connector: WRPConnector that will be used by the camera to communicate with the server
- xml: str, received from the socket

Return

list of instances of `Camera`

CHAPTER 4

WRP Connector

The class WRPConnector is implemented in file wrp_connector.py and has this methods:

class wrpclient.wrp_connector.**WRPConnector**

Finite-state machine that implements WRP and works like a middleman between cameras and driver.

close_camera(*camera_serial_number*, *timeout*)

Moves from state CAMERA_SELECTED to state CONNECTED if the server sends response on the request within the timeout.

Params

- camera_serial_number: str
- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

close_camera_async(*camera_serial_number*)

Asynchronously moves from state CAMERA_SELECTED to state CONNECTED.

Params

- camera_serial_number: str

Return

None

connect(*ip_address*, *port*, *timeout*)

Moves from state IDLE to state CONNECTED if the connection with the server is established within the timeout.

Params

- ip_address: str, IP address of the WRP server
- port: int, port of the WRP server

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

connect_async(*ip_address*, *port*)
Asynchronously moves from state IDLE to state CONNECTED.

Params

- ip_address: str, IP address of the WRP server
- port: int, port of the WRP server

Return

None

disconnect(*timeout*)
Moves from state CONNECTED to state IDLE if the server confirms the request within the timeout.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

disconnect_async()
Asynchronously moves from state CONNECTED to state IDLE. **Params**

None

Return

None

get_cameras(*timeout*)
Moves from state CONNECTED to state WAITING_FOR_CAMERA_LIST and back if the server sends response on the request within the timeout.

Params

- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

get_cameras_async()
Asynchronously moves from state CONNECTED to state WAITING_FOR_CAMERA_LIST and back.

Params

None

Return

None

get_frame(*camera_serial_number*, *timeout*)
Moves from state CAMERA_SELECTED to state WAITING_FOR_FRAME and back if the server sends response on the request within the timeout.

Params

- camera_serial_number: str
- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

get_frame_async(*camera_serial_number*)
Asynchronously moves from state CAMERA_SELECTED to state WAITING_FOR_FRAME and back.

Params

- camera_serial_number: str

Return

None

is_camera_open(*camera_serial_number*)
Check if the camera with a given serial number is open.

Params

- camera_serial_number: str

Return

bool

is_connected()
Check if connection was established

Params

None

Return

bool

open_camera(*camera_serial_number*, *timeout*)
Moves from state CONNECTED to state CAMERA_SELECTED if the server sends response on the request within the timeout.

Params

- camera_serial_number: str
- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

open_camera_async(*camera_serial_number*)
Asynchronously moves from state CONNECTED to state CAMERA_SELECTED.

Params

- camera_serial_number: str

Return

None

start_continuous_shot(*camera_serial_number*, *callback*, *timeout*)
Moves from state CAMERA_SELECTED to state CONTINUOUS_GRABBING if the server sends response on the request within the timeout.

Params

- camera_serial_number: str
- callback: callable
- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

start_continuous_shot_async(*camera_serial_number*)
Asynchronously moves from state CAMERA_SELECTED to state CONTINUOUS_GRABBING.

Params

- camera_serial_number: str

Return

None

stop_continuous_shot(*camera_serial_number*, *timeout*)
Moves from state CONTINUOUS_GRABBING to state CAMERA_SELECTED if the server sends response on the request within the timeout.

Params

- camera_serial_number: str
- timeout: int, time in seconds that is given to the server to response until TimeoutError exception is raised

Return

None

stop_continuous_shot_async(*camera_serial_number*)
Asynchronously moves from state CONTINUOUS_GRABBING to state CAMERA_SELECTED.

Params

- camera_serial_number: str

Return

None

CHAPTER 5

Workswell Remote Protocol (WRP)

State diagram of WRP is following:

Black arrows are events trigggered by the client, red ones comes from the server. Message is composed of header, that specifies message type, payload length and payload. Messages are:

Type	Type value	Payload size	Payload
INVALID	0	0	
OK	1	0	
ERROR	2	1	Error code
GET_CAMERA_LIST	3	0	
CAMERA_LIST	4	variable	XML with listed cameras
OPEN_CAMERA	5	variable	Serial no. of camera
CLOSE_CAMERA	6	0	
GET_FRAME	7	0	
FRAME	8	variable	Frame no., Timestamp, Height, Width, Frame
START_CONTINUOUS_GRABBING	9	0	
STOP_CONTINUOUS_GRABBING	10	0	
ACK_CONTINUOUS_GRABBING	11	5	Frame no.

Payload content datatypes:

Name	Datatype
Error code	uint8
XML with listed cameras	string
Serial no. of cameras	string
Camera ID	uint8
Frame no.	uint32
Timestamp	uint64
Height	uint16
Width	uint16
Frame	array of 32-bit float

Error codes are:

Code	Code value
UNEXPECTED_MESSAGE	0
CAMERA_NOT_FOUND	1
CAMERA_NOT_RESPONDING	2
CAMERA_NOT_OPEN	3
CAMERA_NOT_CONNECTED	4
CAMERA_NOT_ACQUIRING	5

CHAPTER 6

Installation

The simplest way to install *WRP* client is from the pypi:

```
pip install wrpclient
```

Alternative method is to build this repository:

```
git clone https://github.com/Kasape/wrpclient.git
cd wrpclient
python setup.py install
```

CHAPTER 7

Usage

This project is implemented using asyncio library. But because using asyncio library can be a little problematic for beginners in Python, there are also synchronous wrappers about the asynchronous ones. First we have to create a instance of Client class and then connect it to the server:

```
from datetime import datetime
import wrp_client
import asyncio

client = wrp_client.Client()
SERVER_IP_ADDRESS = "127.0.0.1"
# synchronous wrapper for the method (coroutine)
# client.connect_async(ip_adress=SERVER_IP_ADDRESS)
client.connect(ip_adress=SERVER_IP_ADDRESS, timeout="20")
```

Once the client is connected to the server, we can get list of all cameras that was identified by the server.

```
# get all cameras
all_cameras = client.get_cameras(timeout="20")
```

Or we can get only one camera identified by the serial number. If the camera is not available, ValueError exception is raised. Then we have to open the camera to get frame(s):

```
# find camera with specific serial number
my_camera = client.get_camera(serial_number="ABCDEF", timeout="20")

my_camera.open(timeout="20")

# Return 2D frame (numpy matrix) with dtype np.float32 filled with raw data (decimal␣
↪values of temperatures)
frame = my_camera.get_frame(timeout="20")
```

As you can see, all the functions above have parameter timeout. That is because each function is sending some request and is expecting response from to the server and latence of the server depends on the latence of the cameras. There are also asynchronous versions of these functions for more advanced users. They are named *xxx_async* as shown in case of *client.connect_async*.

You can also ask camera for the continuous stream of frames:

```
def callback(frame):
        time_str = datetime.now().strftime("%Y-%m-%d-%H-%M-%S-%f")
        frame_color = cv2.applyColorMap(frame, cv2.COLORMAP_JET)
        cv2.imwrite(f"frame-{time_str}.jpg", frame_color)

# give handler for continuous shot that saves colorized images with timestamp suffix
my_camera.start_continuous_shot(callback)

# wait some time to collect images
asyncio.sleep(5)

my_camera.stop_continuous_shot(callback)
```

If you want to use the API in IPython enviroment (most common are Jupyter notebooks), you have to install *Nest asyncio <https://pypi.org/project/nest-asyncio/>* and run the following code before using wrpclient:

```
import nest_asyncio
nest_asyncio.apply()
```

CHAPTER 8

Documentation

Above you can find guide for installation and example of usage. The full version of the documentation also containing class and methods description (API) can be found on ReadTheDocs page or you can build it from a repository with code:

```
git clone https://github.com/Kasape/wrpclient.git
cd wrpclient/docs
pip install sphinx
make html
```

and open it with your browser on the address `file://<path_to_repo>/docs/_build/html/index.html`.

CHAPTER 9

Licence

This project has GNU GPLv3 License.

Python Module Index

W

Index

S

W

X