aquaipy Documentation

Release 2.0.1

Author

Jun 11, 2019

Contents:

CHAPTER 1

Introduction

AquaIPy exposes the functionality that is usually available, via the AquaIllumination mobile phone app or web app, for the Prime HD and Hydra HD ranges of aquarium lights. AquaIPy is written with full async support, using asyncio but also provides synchronous wrappers for all functions.

This has been tested and validated with a Prime HD and a Hydra 26HD. It should work with a Hydra 52HD but I don't own one to test against, contact me if you have one and you are willing to help me validate the library. In theory it can be extended to support the non-HD models but I haven't had access to those models for testing yet. There is some code in place, that may handle them but I would again need access to them to validate. If you have one, and are willing to help with testing, let me know.

Note: I have tried my best to validate, and test, this functionality to make sure there were no issues but using this could, and most likely will, invalidate the warranty that AquaIllumination provides, and could do damange to your light. Using this library is entirely at your own risk. This software is provided, *AS IS*, without any warranties or conditions of any kinds. I will not be held responsible for any damages. This library is released under the Apache Version 2.0 license, which states the same and a copy can be found with the source code.

1.1 Basics

Here are some basic details of how to use this library.

1.1.1 Install

The easiest method to install `AquaIPy` is using `pip`.:

```
$ pip install AquaIPy
```

1.1.2 Connecting to your AI light

Connecting to your AI light is simple. For the case where your AI light is accessible at the IP `192.168.1.10`, all that is required is.:

```
>>> from aquaipy import AquaIPy
>>> ai = AquaIPy()
>>> await ai.async_connect("192.168.1.10")
```

This will verify connectivity to the light and check the firmware version is supported. In the case where you have multiple lights paired together, AquaIPy expects to be connected to the parent/primary light, of the group and will give an error if it is connected to one of the child lights.

1.1.3 Getting/Setting the schedule state

Working off the assumption that the majority of lights will be currently running on a schedule already, it's important to be able to disable and enable the schedule. If you don't disable the schedule and try to set the colors with the library, the colors will only change for a second and then change back to the schedule.

Getting the schedule state is easy.:

```
>>> await ai.async_get_schedule_state()
True
```

Setting it is easy as well.:

```
>>> await ai.async_set_schedule_state(False)
<Response.Success: 0>
```

1.1.4 Playing around with color

The AI lights provide a number of color channels. The `AquaIPy` library provides a range of different functions to manage colors.

Getting the colors

The API will only accept certain colors, you can get the `list` of valid colors with the following call.:

```
>>> await ai.async_get_colors()
['deep_red', 'royal', 'cool_white', 'violet', 'green', 'blue', 'uv']
```

It's also possible to retrieve the list of the colours and their current state with the following call.:

```
>>> await ai.async_get_colors_brightness()
{'blue': 18.7,
 'cool_white': 4.4,
 'deep_red': 1.0,
 'green': 1.3,
 'royal': 18.4,
 'uv': 46.3,
 'violet': 46.8}
```

Setting the colors

The API also provides a number of different ways to set, patch all colors, as well as increase/decrease a single color channel.

All colors can be set in one call to the function below, by providing a `dict` of colors and their percentage value.:

```
>>> await ai.async_set_colors_brightness(all_colors)
<Response.Success: 0>
```

It also possible to modify only a subset of the colors by providing them as a `dict`.:

```
>>> await ai.async_patch_colors_brightness(subset_colors)
<Response.Success: 0>
```

Lastly, it's possible to update a given color channel by a specified percentage.:

```
>>> await ai.async_update_color_brightness('cool_white', 33.333)
<Response.Success: 0>
>>> await ai.async_update_color_brightness('deep_red', -15.2)
<Response.Success: 0>
```

Response Codes

The library can return a number of response codes as `Response` objects. The respone codes that you should be aware of are below. If any of the these error response codes are returned, then the call will have failed and no changes will have been made:

- `Response.AllColorsMustBeSpecified` - returned when a call to `async_set_colors_brightness()` doesn't include all colors.
- `Response.PowerLimitExceeded` - returned when a call to one of the methods that updates the colors, would have exceeded the max wattage allowed for the targeted light.
- `Response.InvalidData` - returned when invalid data is supplied to one of the methods that updates the colors.

1.2 API

AquaiPy provides an API for the AquaIllumination range of lights.

1.2.1 aquaipy.aquaipy module

Module for working with the AquaIllumination range of lights.

class aquaipy.aquaipy.**AquaIPy**(*name=None*, *session=None*, *loop=None*)
Bases: object

A class that exposes the AquaIllumination Lights API.

async_connect(*host*, *check_firmware_support=True*)
Connect **AquaIPy** instance to a specified AI light.

Also verifies connectivity and firmware version support.

Parameters

- **host** – Hostname/IP of AI light, for paired lights this should be the parent light.
- **check_firmware_support** (*bool*) – Set to False to skip the firmware check

Note: It is **NOT** recommended to set *check_firmware_support=False*. Do so at your own risk!

Raises

- ***FirmwareError*** – If the firmware version is unsupported.
- ***ConnError*** – If unable to connect to specified AI light.
- ***MustBeParentError*** – the specified host must be the parent light, if there are multiple lights linked.

Example

```
>>> from aquaipy import AquaIPy
>>> ai = AquaIPy()
>>> await ai.async_connect("192.168.1.1")
```

async_get_colors()
Get the list of valid colors to pass to other colors methods.

Returns list of valid colors or *None* if there's an error

Return type list(color_1..color_n) or None

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `async_connect()` has failed

async_get_colors_brightness()
Get the current brightness of all color channels.

Returns dictionary of color and brightness percentages, or *None* if there's an error

Return type dict(color_1=percentage_1..color_n=percentage_n) or None

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

async_get_schedule_state()
Check if light schedule is enabled/disabled.

Returns Schedule Enabled (*True*) / Schedule Disabled (*False*) or *None* if there's an error

Return type bool

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

async_patch_colors_brightness(*colors*)
Set specified colors to the given percentage brightness.

Parameters **colors** (*dict(color_1=percentage_1..color_n=percentage_n)*) – Specify just the colors that should be updated

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

async_set_colors_brightness(*colors*)
Set all colors to the specified color percentage.

Note: All colors returned by *get_colors()* must be specified.

Parameters **colors** (*dict(color_1=percentage_1..color_n=percentage_n)*) – dictionary of colors and percentage values

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

async_set_schedule_state(*enable*)
Enable/disable the light schedule.

Parameters **enable** (*bool*) – Schedule Enable (*True*) / Schedule Disable (*False*)

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

async_update_color_brightness(*color*, *value*)
Update a given color by the specified brightness percentage.

Parameters

- **color** (*str*) – color to change
- **value** (*float*) – value to change percentage by

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed.

base_path
Get base path of the AI API.

Returns base path

Return type str

close()
Clean-up and close the underlying async dependancies..

Note: There is no async method, as it is assumed if you are using async functions, you will use your own event loop and aiohttp.ClientSession and pass them in. This will close the client session and event loop, if they were created by this object, when it was initialised.

connect(*host*, *check_firmware_support=True*)
Connect **AquaIPy** instance to a specifed AI light, synchronously.

Parameters

- **host** – Hostname/IP of AI light, for paired lights this should be the parent light.
- **check_firmware_support** (*bool*) – Set to False to skip the firmware check

Note: It is **NOT** recommended to set *check_firmware_support=False*. Do so at your own risk!

Raises

- ***FirmwareError*** – If the firmware version is unsupported.
- ***ConnError*** – If unable to connect to specified AI light.
- ***MustBeParentError*** – the specified host must be the parent light, if there are multiple lights linked.

Example

```
>>> from aquaipy import AquaIPy
>>> ai = AquaIPy()
>>> ai.connect("192.168.1.1")
```

firmware_version
Get firmware version.

Returns firmware version

Return type str

get_colors()
Get the list of valid colors for other methods, synchronously.

Returns list of valid colors or *None* if there's an error

Return type list(color_1..color_n) or None

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

get_colors_brightness()
Get the current brightness of all color channels, synchronously.

Returns dictionary of color and brightness percentages, or *None* if there's an error

Return type dict(color_1=percentage_1..color_n=percentage_n) or None

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

get_schedule_state()
Check if light schedule is enabled/disabled, synchronously.

Returns Schedule Enabled (*True*) / Schedule Disabled (*False*) or *None* if there's an error

Return type bool

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

mac_addr
Get connected devices Mac Address/Serial Number.

Returns device mac address/serial number

Return type str

name
Get device name.

Returns device name

Return type str

patch_colors_brightness(*colors*)
Set specified colors to the given percentage values, sychronously.

Parameters **colors** (*dict(color_1=percentage_1..color_n=percentage_n)*) – Specify just the colors that should be updated

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

product_type
Get product type.

Returns product type

Return type str

set_colors_brightness(*colors*)
Set all colors to the specified color percentage, synchronously.

Parameters **colors** (*dict(color_1=percentage_1..color_n=percentage_n)*) – dictionary of colors and percentage values

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

set_schedule_state(*enable*)
Enable/Disable the light schedule, synchronously.

Parameters **enable** (*bool*) – Schedule Enable (*True*) / Schedule Disable (*False*)

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed

supported_firmware
Check if current firmware is supported.

Returns status of firmware support

Return type bool

update_color_brightness(*color*, *value*)
Update a given color by the specified brightness, synchronously.

Parameters

- **color** (*str*) – color to change

- **value** (*float*) – value to change percentage by

Returns Response.Success if it works, or a value indicating the error, if there is an issue.

Return type *Response*

Raises ***ConnError*** – if there is no valid connection to a device, usually because a previous call to `connect()` has failed.

class aquaipy.aquaipy.**HDDevice**(*raw_data*, *primary_mac_address=None*)

Bases: object

A class for handling the conversion of data for a device.

convert_to_intensity(*color*, *percentage*)

Convert a percentage to the native AI API intensity value.

The conversion is different for every color and model of light. The intensity to be returned will be bewtween 0-1000 for non-HD values and between 1000-2000 for HD (over 100%) values.

Parameters

- **color** (*str*) – the specified color to convert
- **percentage** (*float*) – the percentage to convert

Returns intensity value (0-2000)

Return type int

convert_to_mw(*color*, *intensity*)

Convert a given AI API native intensity value to the mWatt value.

An input intesnity (0-2000), is converted to the equivalent mWatt value for that color channel, on the specified device. This will differ for each color channel and device type.

Parameters

- **color** (*str*) – the specified color to convert
- **intensity** (*int*) – the specified color intensity (0-2000)

Returns the resulting mWatt value, for the given intensity

Return type float

convert_to_percentage(*color*, *intensity*)

Convert the native AI API intensity value to a percentage.

The conversion of the intensity value (0 - 2000) will be different for every color and light model.

Parameters

- **color** (*str*) – the specified color to convert
- **intensity** (*int*) – the specified color intensity (0-2000)

Returns the resulting percentage

Return type float

is_primary

Check is HDDevice object represents a parent light.

Returns true if parent, false if not

Return type bool

mac_address
Get devices MAC address/serial number.

Returns MAC address of device

Return type str

max_mw
Get the max mWatts supported power level for the device.

Returns max mWatts

Return type int

class aquaipy.aquaipy.**Response**
Bases: enum.Enum

Response codes, for the AquaIPy methods.

AllColorsMustBeSpecified = 5

Error = 1

InvalidBrightnessValue = 3

InvalidData = 6

NoSuchColour = 2

PowerLimitExceeded = 4

Success = 0

1.2.2 aquaipy.error module

This modules contains all the errors that are part of AquaIPy.

exception aquaipy.error.**ConnError**(*message*, *host*)
Bases: *aquaipy.error.Error*

Raised when there is a connection error for an AI light.

Variables

- **message** – error message
- **host** – host

exception aquaipy.error.**Error**
Bases: Exception

Base class for exceptions in this class.

exception aquaipy.error.**FirmwareError**(*message*, *firmware_version*)
Bases: *aquaipy.error.Error*

Raised when connecting to a device that has unsupported firmware.

Variables

- **message** – error message
- ***firmware_version*** – unsupported version of firmware

exception aquaipy.error.**MustBeParentError**(*message*, *parent_identifier*)

Bases: *aquaipy.error.Error*

Raised when connecting to a light that isn't the parent.

The AquaIPy library only supports connecting to a parent light. There is support for paired devices, as long as the parent is the device that is connected to.

Variables

- **message** – error message
- **parent_identifier** – an identifier for the parent device

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

R

S

U