aioswitcher

Release 1.2.1

Tomer Figenblat

Jul 02, 2020

CONTENTS

CHAPTER

ONE

INSTALL

```
pip install aioswitcher
```

That's it!

Note:

- `Switcher V2` was verified by the author of this project.
- `Switcher Touch` has been verified by various users of this project.

CHAPTER

TWO

PREREQUISITES

- Install and configure your Switcher device.
- Collect the following information from the device's following NightRang3r instructions in the Switcher-V2-Python repository:
- ip_address
- phone_id
- device_id
- device_pass

CHAPTER

THREE

USAGE

This module provides two separate integrations with the Switcher device, both fully asynchronous.

The first integration is a *UDP Bridge* constantly listening to broadcast messages launched from the device approximately every 4 seconds. This integration provides constant device state updates.

The second integration is a *TCP Socket API* providing the ability, to not only get the current state of the device, but also control it.

TOC

3.1 UDP Bridge

With the *UDP Bridge* integration, we're constantly listening to messages broadcast from the device containing information about the device and the device's state.

The message is being broadcast from the device approximately every 4 seconds, once the message is received and verified it will be `put` in a asyncio.Queue with the max size of 1, which means the `Queue` always has `1` or `None` messages waiting, containing the latest state received.

Each message will be an updated instance of the *SwitcherV2Device* object.

The bridge can be used as a `Context Manager` as well as being instantiated and controlled manually.

Note: The *UDP Bridge* integration will allow you to receive **Real-Time** updates from the device approximately every 4 seconds, yet it will not allow you to control the device.

For controlling the device you can use the *TCP Socket API* integration. (Both work simultaneously).

3.1.1 Example of UDP Bridge usage

```
import asyncio
from datetime import datetime
from aioswitcher.bridge import SwitcherV2Bridge
from aioswitcher.devices import SwitcherV2Device

# instructions on getting this data is in,
# https://github.com/NightRang3r/Switcher-V2-Python
phone_id = "your_devices's_phone_id"
device_id = "your_devices's_device_id"
device_password = "your_devices's_device_password"

# create a new event loop
your_loop = asyncio.get_event_loop()

"""Use as instance."""
async def run_as_instance() -> None:
    v2bridge = SwitcherV2Bridge(
        your_loop, phone_id, device_id, device_password
    )
    # start the bridge
    await v2bridge.start()

    # get the Queue
    queue = v2bridge.queue  # type: asyncio.Queue

    # create an event to signal your coroutine to start/stop
    signal_event = asyncio.Event()
    signal_event.set()

    # coroutine for getting the device from the queue
    async def get_device_from_queue() -> None:
        device = await queue.get()  # type: SwitcherV2Device
        print("instance state is: {}".format(device.state))
        print(datetime.now())
        if signal_event.is_set():
```

(continues on next page)

(continued from previous page)

```
            your_loop.call_soon(get_device_from_queue)
        return None

    # schedule your coroutine which will wait for the device
    # to be put in the queue, and print the time and its state
    # afterwards it will call itself again, the result will be
    # the device state being printed every approximately 4 seconds
    your_loop.call_soon(get_device_from_queue)

    # wait for 40 seconds
    # the state should be printed about 8 to 10 times
    await asyncio.sleep(40)

    # stop the coroutine
    signal_event.clear()

    # stop the bridge
    await v2bridge.stop()

    return None

"""Use as context manager."""
async def run_as_context_manager() -> None:
    async with SwitcherV2Bridge(
        your_loop,
        phone_id,
        device_id,
        device_password,
    ) as v2bridge:
        # get the Queue
        queue = v2bridge.queue  # type: asyncio.Queue

        # create an event to signal your coroutine to start/stop
        signal_event = asyncio.Event()
        signal_event.set()

        # coroutine for getting the device from the queue
        async def get_device_from_queue() -> None:
            device = await queue.get()  # type: SwitcherV2Device
            print("context manager state is: {}".format(device.state))
            print(datetime.now())
            if signal_event.is_set():
                your_loop.call_soon(get_device_from_queue)
            return None

        # schedule your coroutine which will wait for the device
        # to be put in the queue, and print the time and its state
        # afterwards it will call itself again, the result will be
        # the device state being printed every approximately 4 seconds
        your_loop.call_soon(get_device_from_queue)

        # wait for 40 seconds
        # the state should be printed about 8 to 10 times
        await asyncio.sleep(40)

        # stop the coroutine
        signal_event.clear()
```

(continues on next page)

(continued from previous page)

```
    return None

your_loop.run_until_complete(run_as_instance())
your_loop.run_until_complete(run_as_context_manager())

loop.close()
```

3.2 TCP Socket API

With *TCP Socket API* integration we gain the following abilities:

- Get the device status
- Control the device
- Get the schedules from the device
- Set the device name
- Set the device Auto-Off configuration
- Create/Delete/Enable/Disable schedules on the device.

Note: Although the *TCP Socket API* is applicable as a `context manager` and as an instance of an object, It is preferable to use it as a `context manager` due to the nature of the `tcp connection` (you don't want to occupy a connection slot on the device any longer then you have to or you'll start seeing `TimeOutErrors`).

To use as an instance (which will not be covered here), you can rely on the `UDP Bridge` example and just substitute `start()` and `stop()` with `connect()` and `disconnect()`.

The various responses are covered in the *API Response Messages* section.

3.2.1 Example of TCP Socket API usage

```
import asyncio
from datetime import timedelta
from aioswitcher import consts, tools
from aioswitcher.api import SwitcherV2Api, messages
from aioswitcher.schedules import SwitcherV2Schedule

# create a new event loop
your_loop = asyncio.get_event_loop()

# if you're also using the udp bridge,
# the ip address is available at (SwitcherV2Device).ip_addr
ip_address = "your_device's_ip_address"

# instructions on getting this data is in
# https://github.com/NightRang3r/Switcher-V2-Python
phone_id = "your_devices's_phone_id"
device_id = "your_devices's_device_id"
device_password = "your_devices's_device_password"

"""Use as context manager."""
async def run_as_context_manager() -> None:
```

(continues on next page)

(continued from previous page)

```
    async with SwitcherV2Api(
            your_loop, ip_address, phone_id,
            device_id, device_password) as swapi:
        # get the device state
        # response: messages.SwitcherV2StateResponseMSG
        state_response = await swapi.get_state()

        # control the device: on / off / on + (15/30/45/60) minutes timer
        # response: messages.SwitcherV2ControlResponseMSG
        turn_on_response = await swapi.control_device(
            consts.COMMAND_ON)
        turn_off_response = await swapi.control_device(
            consts.COMMAND_OFF)
        turn_on_30_min_response = await swapi.control_device(
            consts.COMMAND_ON, '30')

        # set the limit time to auto-shutdown the device (1 < hours < 24)
        # response: messages.SwitcherV2SetAutoOffResponseMSG
        time_to_off = timedelta(hours=1, minutes=30)
        set_auto_off_response = await swapi.set_auto_shutdown(time_to_off)

        # set the device name (2 < length < 33)
        # response: messages.SwitcherV2UpdateNameResponseMSG
        set_name_response = await swapi.set_device_name("new device name")

        # get the configured schedules from the device
        # response: messages.SwitcherV2GetScheduleResponseMSG
        get_schedules_response = await swapi.get_schedules()

        # disable or enable a schedule
        # schedule_data = (SwitcherV2Schedule).schedule_data
        # response: messages.SwitcherV2DisableEnableScheduleResponseMSG
        #
        # the following will enable the schedule:
        # updated_schedule_data = (
        #    schedule_data[0:2] + consts.ENABLE_SCHEDULE + schedule_data[4:])
        #
        # the following will disable the schedule:
        # updated_schedule_data = (
        #    schedule_data[0:2] + consts.DISABLE_SCHEDULE + schedule_data[4:])
        enable_disable_response = await swapi.disable_enable_schedule(
            updated_schedule_data)

        # delete a schedule (0 <= schedule_id <= 7)
        # schedule_id = (SwitcherV2Schedule).schedule_id
        # response: messages.SwitcherV2DeleteScheduleResponseMSG
        delete_response = await swapi.delete_schedule(schedule_id)

        # create a schedule to turn on at 20:30 and off at 21:00
        # response: messages.SwitcherV2CreateScheduleResponseMSG
        schedule_days = [0]
        # append selected days, if non-recurring skip next
        schedule_days.append(consts.DAY_TO_INT_DICT[consts.SUNDAY])
        schedule_days.append(consts.DAY_TO_INT_DICT[consts.MONDAY])
        schedule_days.append(consts.DAY_TO_INT_DICT[consts.TUESDAY])
        schedule_days.append(consts.DAY_TO_INT_DICT[consts.WEDNESDAY])
        schedule_days.append(consts.DAY_TO_INT_DICT[consts.THURSDAY])
```

(continues on next page)

(continued from previous page)

```
        schedule_days.append(consts.DAY_TO_INT_DICT[consts.FRIDAY])
        schedule_days.append(consts.DAY_TO_INT_DICT[consts.SATURDAY])
        # skip here if non-recurring
        weekdays = await tools.create_weekdays_value(
            your_loop, schedule_days)
        start_time = await tools.timedelta_str_to_schedule_time(
            your_loop, str(timedelta(hours=20, minutes=30)))
        end_time = await tools.timedelta_str_to_schedule_time(
            your_loop, str(timedelta(hours=21)))
        schedule_data = consts.SCHEDULE_CREATE_DATA_FORMAT.format(
            weekdays, start_time, end_time)
        create_response = await swapi.create_schedule(schedule_data)

    return None

your_loop.run_until_complete(run_as_context_manager())

your_loop.close()
```

3.3 Objects and Properties

There are two main objects you need to be aware of:

- The first object is the one representing the device itself, `aioswitcher.devices.SwitcherV2Device` *SwitcherV2Device*.
- The second object is the one representing the device's schedule, `aioswitcher.schedules.SwitcherV2Schedule` *SwitcherV2Schedule*.

3.3.1 SwitcherV2Device

Property	Type	Description	Possible Values	Default Value
device_id	`str`	Return the device id.	ab1c2d	
ip_addr	`str`	Return the ip address.	192.168.100.157	waiting_for_data
mac_addr	`str`	Return the mac address.	A1:B2:C3:45:67:D8	waiting_for_data
name	`str`	Return the device name.	device name	waiting_for_data
state	`str`	Return the device state.	on, off	
remaining_time	`str`	Return the auto-off configuration value.	%H:%M:%S	waiting_for_data
auto_off_set	`str`	Return the time left to auto-off.	%H:%M:%S	waiting_for_data
power_consumption	`int`	Return the power consumption in watts.	2780	0
electric_current	`float`	Return the power consumption in amps.	12.8	0.0
phone_id	`str`	Return the the phone id.	1234	
last_data_update	`datetime`	Return the timestamp of the last update.	%Y-%m-%dTH:%M:%S.%F	
last_state_change	`datetime`	Return the timestamp of the last state change.	%Y-%m-%dTH:%M:%S.%F	

3.3.2 SwitcherV2Schedule

Property	Type	Description	Possible Values	Default
schedule_id	`str`	Return the schedule id.	0-7	
enabled	`bool`	Return true if enabled. Has a setter manipulating the schedule status.	True, False	False
recurring	`bool`	Return true if recurring.	True, False	False
days	`List[str]`	Return the weekdays of the schedule.	• Sunday • Monday • Tuesday • Wednesday • Thursday • Friday • Saturday • **Every day**	
start_time	`str`	Return the start time of the schedule.	%H:%M	waiting_for_data
end_time	`str`	Return the end time of the schedule.	%H:%M	waiting_for_data
duration	`str`	Return the duration time of the schedule.	0:30:00	waiting_for_data
schedule_data	`str`	Return the schedule data for managing the schedule. has a setter manipulating the schedule data.	Any	waiting_for_data
init_future	`asyncio.Future`	Return the future of the init.	SwitcherV2Schedule	

3.4 API Response Messages

The following are the response message objects returning from the various API functions. The source of the responses can be found `aioswitcher.api.messages`.

Please note the `aioswitcher.api.messagesResponseMessageType` *Enum Class* for identifying the response message types:

- *AUTO_OFF*
- *CONTROL*
- *CREATE_SCHEDULE*
- *DELETE_SCHEDULE*
- *DISABLE_ENABLE_SCHEDULE*
- *GET_SCHEDULES*

- *STATE*
- *UPDATE_NAME*

3.4.1 SwitcherV2BaseResponseMSG

Property	Type	Description
unparsed_response	`bytes`	Return the unparsed response message.
successful	`bool`	Return the status of the message.
msg_type	`ResponseMessageType`	Return the response message type.

3.4.2 SwitcherV2StateResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.STATE`

Property	Type	Description
state	`str`	Return the state. Possible values are: • `aioswitcher.consts.STATE_ON` • `aioswitcher.consts.STATE_OFF`
time_left	`str`	Return the time left to auto-off.
time_on	`str`	Return the time in "on" state. Relevant only if the current state is "on".
auto_off	`str`	Return the auto-off configuration value.
power	`Optional[int]`	Return the current power consumption in watts.
current	`Optional[float]`	Return the power consumption in amps.
init_future	`asyncio.Future`	Return the future of the initialization. As the initialization of this message requires some asynchronous actions, please use `init_future.result()` to get the message object.

3.4.3 SwitcherV2ControlResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.CONTROL`

```
No properties are added by object.
```

3.4.4 SwitcherV2SetAutoOffResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.AUTO_OFF`

```
No properties are added by object.
```

3.4.5 SwitcherV2UpdateNameResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.UPDATE_NAME`

```
No properties are added by object.
```

3.4.6 SwitcherV2GetScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.GET_SCHEDULES`

Property	Type	Description
found_schedules	`bool`	Return true if found schedules in the response.
get_schedules	`List[SwitcherV2Schedule]`	Return a list of *SwitcherV2Schedule*.

3.4.7 SwitcherV2DisableEnableScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.DISABLE_ENABLE_SCHEDULE`

```
No properties are added by object.
```

3.4.8 SwitcherV2DeleteScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.DELETE_SCHEDULE`

```
No properties are added by object.
```

3.4.9 SwitcherV2CreateScheduleResponseMSG

Extends *SwitcherV2BaseResponseMSG*

Response Type `ResponseMessageType.CREATE_SCHEDULE`

```
No properties are added by object.
```

CHAPTER

FOUR

LICENSE

CHAPTER

FIVE

CREDITS

This project wouldn't have been able to happen without the amazing work preformed by NightRang3r and AviadGolan in the Switcher-V2-Python project.

So... Thanks!

CHAPTER

SIX

PROJECTS USING THIS MODULE

- Home Assistant's `switcher_kis` component:
  - switcher_kis code
  - switcher_kis documentation
- Switcher Water Heater Unofficial REST API:
  - webapi code
  - webapi documentation

Note: If you use this module in your open-source project and you want it to be listed here, please let me know (or add it yourself via github).

CHAPTER

SEVEN

CODE DOCUMENTATION

7.1 aioswitcher

Switcher water heater unofficial API and bridge.

7.2 aioswitcher.api

Switcher water heater unofficial API and bridge, API Object.

class aioswitcher.api.**SwitcherV2Api**(*loop: asyncio.events.AbstractEventLoop*, *ip_addr: str*, *phone_id: str*, *device_id: str*, *device_password: str*)

Represntation of the SwitcherV2 API object.

Parameters

- **loop** – the event loop for the factory to run in.
- **ip_addr** – the ip address assigned to the device
- **phone_id** – the phone id of the desired device.
- **device_id** – the id of the desired device.
- **device_password** – the password of the desired device.

Todo:

- control_device takes a timer value that must be converted to hex before calling this method using the deisgnated tool. On the other hand, the rest of the action methods` takes their arguments raw and use the appropriate tool for converting themselves. This is confusing and needs to be adjusted.

---

async __aenter__()

Enter SwitcherV2Api asynchronous context manager.

Returns This instance of aioswitcher.api.SwitcherV2Api as an awaitable.

async __await__()

Return SwitcherV2Api awaitable object.

Returns This instance of aioswitcher.api.SwitcherV2Api.

async __aexit__(*exc_type*, *exc_value*, *traceback*)

Exit SwitcherV2Api asynchronous context manager.

async connect()
Connect to asynchronous socket and get reader and writer object.

async disconnect()
Disconnect from asynchronous socket.

async _full_login()
Use for sending the login packet to the device.

Returns

A tuple of two `Tuple[str, Optional[messages.SwitcherV2LoginResponseMSG]]`.

The first object is a string contianing the hexadecimal representation of the current unix timestamp.

The second object will be An instance of the serialized object `aioswitcher.api.messages.SwitcherV2LoginResponseMSG`.

Note: This is a private function, please consider using its wrapper function `login` instead.

async login()
Use as wrapper for sending the login packet to the device.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2LoginResponseMSG`.

async _full_get_state()
Use for sending the get state packet to the device.

Returns

A tuple of three `Tuple[str, Optional[messages.SwitcherV2LoginResponseMSG], Optional[messages.SwitcherV2StateResponseMSG]]`.

The first object is a string contianing the hexadecimal representation of the current unix timestamp.

The second object will be An instance of the serialized object `aioswitcher.api.messages.SwitcherV2LoginResponseMSG`.

The third object will be An instance of the serialized object `aioswitcher.api.messages.SwitcherV2StateResponseMSG`.

Note: This is a private function, please consider using its wrapper function `get_state` instead.

async get_state()
Use as wrapper for sending the get state packet to the device.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2StateResponseMSG`.

async set_auto_shutdown(*full_time*)
Use for sending the set auto-off packet to the device.

Parameters **full_time** – timedelta value containg the configuration value for auto-shutdown. Accepts anythin between 1 and 3 hours.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG`.

async set_device_name(*name*)
Use for sending the set name packet to the device.

Parameters **name** – string name with the length of 2 >= x >= 32.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG`.

async get_schedules()
Use for retrival of the schedules from the device.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG`.

async disable_enable_schedule(*schedule_data*)
Use for disabling or enabling a schedule on the device.

Parameters **schedule_data** – formatted data for updating the schedule, can be obtained from the `aioswitcher.schedules.SwitcherV2Schedule` object or created using the format `aioswitcher.consts.SCHEDULE_CREATE_DATA_FORMAT` filled with three values: weekdays, start-time and end-time. Weekdays can be created using the tool `aioswitcher.tools.create_weekdays_value`, the start and end times can be created using the `aioswitcher.tools.timedelta_str_to_schedule_time`

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG`.

async delete_schedule(*schedule_id*)
Use for deleting a schedule from the device.

Parameters **schedule_id** – the id of the schedule slot from the device, can be 0-7 as there are 8 slots available. Can be obtained from the `aioswitcher.schedules.SwitcherV2Schedule` object.

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG`.

async create_schedule(*schedule_data*)
Use for creating a new schedule in the next empty schedule slot.

Parameters **schedule_data** – formatted data for updating the schedule, can be created using the format `aioswitcher.consts.SCHEDULE_CREATE_DATA_FORMAT` filled with three values: weekdays, start-time and end-time. Weekdays can be created using the tool `aioswitcher.tools.create_weekdays_value`, the start and end times can be created using the `aioswitcher.tools.timedelta_str_to_schedule_time`

Returns An instance of the serialized object `aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG`.

property connected
Return true if api is connected.

Type bool

7.3 aioswitcher.api.messages

Switcher Packet Response Messages.

class aioswitcher.api.messages.**ResponseMessageType**(*value*)
Bases: enum.Enum

An enumeration.

class aioswitcher.api.messages.**SwitcherV2BaseResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*, *msg_type:* aioswitcher.api.messages.ResponseMessageType)

Represntation of the switcher v2 base response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.
- **msg_type** – the message type as described in the ''ResponseMessageType" Enum class.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2ControlResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)
Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 control response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2CreateScheduleResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 create schedule response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2DeleteScheduleResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 delete schedule response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2DisableEnableScheduleResponseMSG**(*loop: asyncio.events.AbstractEvent re- sponse: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 dis/en schedule response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2GetScheduleResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

represnation of the switcher v2 get schedule message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

Todo:

- the `get_schedules` attribute should be a method.
- `schdule_detais` is `__init__` is yielding List[str] instead of List[bytes], that's not supposed to happen.

---

property found_schedules
Return true if found schedules in the response.

Type bool

property get_schedules
Return schedules.

Type list(*aioswitcher.schedules.SwitcherV2Schedule*)

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2LoginResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 login response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property session_id
Return the retrieved session id.

Type str

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2SetAutoOffResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 set auto-off response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2StateResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 state response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

Todo:

- replace `init_future` attribute with `get_init_future` method.

---

async initialize(*response*)
Finish the initialization of the message and update the future object.

Parameters **response** – the raw response from the device.

property auto_off
Return the auto-off configuration value.

Type str

property current
Return the power consumption in amps.

Type float

property init_future
Return the future of the initialization.

Type asyncio.Future

async initialize(*response: bytes*) → None
Finish the initialization of the message and update the future object.

Parameters **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property power
Return the current power consumption in watts.

Type int

property state
Return the state.

Type str

property successful
Indicating whether or not the request was successful.

Type bool

property time_left
Return the time left to auto-off.

Type str

property time_on
Return the time in "on" state.

Relevant only if the current state is "on".

Type str

property unparsed_response
Return The raw response from the device.

Type bytes

class aioswitcher.api.messages.**SwitcherV2UpdateNameResponseMSG**(*loop: asyncio.events.AbstractEventLoop*, *response: bytes*)

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Represntation of the switcher v2 update name response message.

Parameters

- **loop** – the event loop to perform actions in.
- **response** – the raw response from the device.

property msg_type
the message type.

Type *aioswitcher.api.messages.ResponseMessageType*

property successful
Indicating whether or not the request was successful.

Type bool

property unparsed_response
Return The raw response from the device.

Type bytes

7.4 aioswitcher.api.packets

Switcher water heater unofficial API and bridge, API Packet formats.

7.5 aioswitcher.bridge

Switcher water heater unofficial API and bridge, Bridge Object.

class aioswitcher.bridge.**SwitcherV2Bridge**(*loop: asyncio.events.AbstractEventLoop*, *phone_id: str*, *device_id: str*, *device_password: str*)

Represntation of the SwitcherV2 Bridge object.

Parameters

- **loop** – the event loop for the factory to run in.
- **phone_id** – the phone id of the desired device.

- **device_id** – the id of the desired device.
- **device_password** – the password of the desired device.

Todo:

- replace `queue` attribute with `get_queue` method.

async __aenter__()
Enter SwitcherV2Bridge asynchronous context manager.

Returns This instance of `aioswitcher.bridge.SwitcherV2Bridge` as an awaitable.

async __await__()
Return SwitcherV2Bridge awaitable object.

Returns This instance of `aioswitcher.bridge.SwitcherV2Bridge`.

async __aexit__(*exc_type*, *exc_value*, *traceback*)
Exit the SwitcherV2Bridge asynchronous context manager.

async start()
Create an asynchronous listenr and start the bridge event.

async stop()
Stop the asynchronous bridge.

property queue
Return the queue storing updated device objects.

Type asyncio.Queue

property running
Return true if bridge is running.

Type bool

7.6 aioswitcher.bridge.messages

Switcher Bridge Response Messages.

class aioswitcher.bridge.messages.**SwitcherV2BroadcastMSG**(*loop: asyncio.events.AbstractEventLoop*, *message: Union[bytes, str]*)
Represntation of the SwitcherV2 broadcast message.

Parameters

- **loop** – the event loop to perform actions in.
- **message** – the raw message from the device.

Todo:

- replace `init_future` attribute with `get_init_future` method.

async initialize(*message*)
Finish the initialization and update the future object.

Parameters **message** – the raw message from the device.

property auto_off_set
Return the auto-off configuration value.

Type str

property current
Return the power consumptionin amps.

Type float

property device_id
Return the device id.

Type str

property device_state
Return the state of the device.

Type str

property init_future
Return the future of the device initialization.

Type asyncio.Future

property ip_address
Return the ip address.

Type str

property mac_address
Return the mac address.

Type str

property name
Return the device name.

Type str

property power
Return the power consumptionin watts.

Type int

property remaining_time_to_off
Return the time left to auto-off.

Type str

property verified
Return whether or not the message is a SwitcherV2 message.

Type bool

7.7 aioswitcher.consts

Switcher water heater unofficial API and bridge, Global constants.

7.8 aioswitcher.devices

Switcher water heater unofficial API and bridge, Device classes.

class aioswitcher.devices.**SwitcherV2Device**(*device_id: str*, *ip_address: str*, *mac_address: str*, *name: str*, *state: str*, *remaining_time: Optional[str]*, *auto_off_set: str*, *power_consumption: int*, *electric_current: float*, *phone_id: str*, *device_password: str*, *last_state_change: datetime.datetime*)

Represntation of the switcherv2 device.

Parameters

- **device_id** – the id retrieved from the device.
- **ip_address** – the ip address assigned to the device.
- **mac_address** – the mac address assigned to the device.
- **name** – the name of the device.
- **state** – the current state of the device (in/off).
- **remaining_time** – remaining time (if on).
- **auto_off_set** – configured value for auto shutdown.
- **power_consumption** – the current power consumpstion in watts.
- **electric_current** – the current power consumpstion in amps.
- **phone_id** – the phone id retrieved from the device.
- **device_password** – the password retrieved from the device.
- **last_state_change** – datetime of the last state change.

update_device_data(*ip_address*, *name*, *state*, *remaining_time*, *auto_off_set*, *power_consumption*, *electric_current*, *last_state_change*)

Update the device state and data.

Parameters

- **ip_address** – the ip address assigned to the device.
- **name** – the name of the device.
- **state** – the current state of the device (on/off).
- **remaining_time** – remaining time (if on).
- **auto_off_set** – configured value for auto shutdown.
- **power_consumption** – the current power consumpstion in watts.
- **electric_current** – the current power consumpstion in amps.
- **last_state_change** – datetime of the last state change.

as_dict()
Return as dict.

Returns A dictionary represntation of the object properties. Used to make the object json serializable.

property auto_off_set
Returns the auto-off configuration value.

Type str

property device_id
Returns the device id.

Type str

property device_password
Returns the device password.

Type str

property electric_current
returns the power consumption in amps.

Type float

property ip_addr
Returns the ip address.

Type str

property last_data_update
Returns timestamp of the last update.

Type datetime

property last_state_change
Returns timestamp of the last state change.

Type datetime

property mac_addr
Returns the mac address.

Type str

property name
Returns the device name.

Type str

property phone_id
Returns the phone id.

Type str

property power_consumption
Returns the power consumption in watts.

Type int

property remaining_time
Returns the time left to auto shutdown.

Type str

property state
Returns the device state.

Type str

7.9 aioswitcher.errors

Switcher water heater unofficial API and bridge, Exception classes.

exception aioswitcher.errors.**CalculationError**
Bases: Exception

Exception to be raised when cpu bound calculation is failing.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception aioswitcher.errors.**DecodingError**
Bases: Exception

Exception to be raised when cpu bound decoding is failing.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception aioswitcher.errors.**EncodingError**
Bases: Exception

Exception to be raised when cpu bound encoding is failing.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

7.10 aioswitcher.protocols

Switcher water heater unofficial API and bridge, Network protocols.

class aioswitcher.protocols.**SwitcherV2UdpProtocolFactory**(*loop: asyncio.events.AbstractEventLoop*, *phone_id: str*, *device_id: str*, *device_password: str*, *queue: asyncio.queues.Queue*, *run_factory_evt: asyncio.locks.Event*)
Bases: asyncio.protocols.DatagramProtocol

Represntation of the Asyncio UDP protocol factory.

Parameters

- **loop** – the event loop for the factory to run in.
- **phone_id** – the phone id of the desired device.
- **device_id** – the id of the desired device.
- **device_password** – the password of the desired device.
- **queue** – a asyncio.Queue for the factory to save messages in.

- **asyncio.Event for signaling the factory to run.**
  (*run_factory_evt*) –

Todo:

- replace `factory_future` attribute with `get_factory_future` method.

connection_made(*transport*)
Call on connection established.

datagram_received(*data*, *addr*)
Call on datagram recieved.

error_received(*exc*)
Call on exception recieved.

connection_lost(*exc*)
Call on connection lost.

close_transport(*future*)
Call for closing the transport.

async handle_incoming_messages(*data*, *addr*)
Use for Handling incoming messages.

get_device_from_message(*ip_addr*, *future*)
Use for extracting the device from the broadcast message.

close_transport(*future: _asyncio.Future*) → None
Call for closing the transport.

connection_lost(*exc: Optional[Exception]*) → None
Call on connection lost.

connection_made(*transport: asyncio.transports.BaseTransport*) → None
Call on connection established.

datagram_received(*data: Union[bytes, str]*, *addr: Tuple*) → None
Call on datagram recieved.

error_received(*exc: Optional[Exception]*) → None
Call on exception recieved.

property factory_future
Representing the initialization status.

Type asyncio.Future

get_device_from_message(*ip_addr: str*, *future: _asyncio.Future*) → None
Use for extracting the device from the broadcast message.

async handle_incoming_messages(*data: Union[bytes, str]*, *addr: Tuple*) → None
Use for Handling incoming messages.

pause_writing()
Called when the transport's buffer goes over the high-water mark.

Pause and resume calls are paired – pause_writing() is called once when the buffer goes strictly over the high-water mark (even if subsequent writes increases the buffer size even more), and eventually resume_writing() is called once when the buffer size reaches the low-water mark.

Note that if the buffer size equals the high-water mark, pause_writing() is not called – it must go strictly over. Conversely, resume_writing() is called when the buffer size is equal or lower than the low-water mark. These end conditions are important to ensure that things go as expected when either mark is zero.

NOTE: This is the only Protocol callback that is not called through EventLoop.call_soon() – if it were, it would have no effect when it's most needed (when the app keeps writing without yielding until pause_writing() is called).

resume_writing()
Called when the transport's buffer drains below the low-water mark.

See pause_writing() for details.

7.11 aioswitcher.schedules

Switcher water heater unofficial API and bridge, Schedules.

class aioswitcher.schedules.**SwitcherV2Schedule**(*loop: asyncio.events.AbstractEventLoop*, *idx: int*, *schedule_details: List[bytes]*)
Represnation of the SwitcherV2 schedule slot.

Parameters

- **loop** – the event loop to perform schedule operation in.
- **idx** – the index of the schedule slot (0-7).
- **schedule_details** – the string raw schedule data details.

Todo:

- Replace `init_future` attribute with `get_init_future` method.

---

async initialize(*idx*, *schedule_details*)
Finish the initialization of the schedule.

as_dict()
Return as dict.

Returns A dictionary represntation of the object properties. Used to make the object json serializable.

property days
Return the weekdays of the schedule.

Type list(str)

property duration
Return the duration of the schedule.

Type str

property enabled
Return true if enabled, setter included.

Type bool

property end_time
Return the end time of the schedule.

Type str

property init_future
Return the future of the initialization.

Type asyncio.Future

property recurring
Return true if recurring.

Type bool

property schedule_data
Return the schedule data, setter included.

Type bytes

property schedule_id
Return the schedule id.

Type str

property start_time
Return the start time of the schedule.

Type str

aioswitcher.schedules.**_calc_next_run_for_schedule**(*schedule_details*)
Calculate the next runtime of the schedule.

Parameters **schedule_details** – SwitcherV2Schedule representing the schedule slot.

Returns A pretty string describing the next due run. e.g. "Due tommorow at 17:00".

Note: This is a private function containing blocking code. Please consider using calc_next_run_for_schedule (without the _), to schedule as a task on the event loop.

---

async aioswitcher.schedules.**calc_next_run_for_schedule**(*loop*, *schedule_details*)
Asynchronous wrapper for _calc_next_run_for_schedule.

Use as async wrapper for calling _calc_next_run_for_schedule, calculating the next runtime of the schedule.

Parameters

- **loop** – the event loop to execute the function in.
- **schedule_details** – SwitcherV2Schedule representing the schedule slot.

Returns A pretty string describing the next due run. e.g. "Due tommorow at 17:00".

7.12 aioswitcher.tools

Switcher water heater unofficial API and bridge, Tools and Helpers.

aioswitcher.tools.**_convert_minutes_to_timer**(*minutes: str*) → str
Convert on-timer minutes to hexadecimal before sending.

Parameters **minutes** – on-timer minutes to convert.

Returns Hexadecimal represntation of the mintues argument.

Raises **aioswitcher.erros.EncodingError** – when failed hexlify.

Note: This is a private function containing blocking code. Please consider using `convert_minutes_to_timer` (without the _), to schedule as a task in the event loop.

`aioswitcher.tools.`**`_convert_seconds_to_iso_time`**(*all_seconds: int*) → str

Convert seconds to iso time.

Parameters **`all_seconds`** – the total number of seconds to convert.

Returns %M:%S format. e.g. "02:24:37".

Return type A string represnting the converted iso time in %H

Raises **`aioswitcher.erros.CalculationError`** – when failed to convert the argument.

Note: This is a private function containing blocking code. Please consider using `convert_seconds_to_iso_time` (without the _), to schedule as a task in the event loop.

`aioswitcher.tools.`**`_convert_string_to_device_name`**(*name: str*) → str

Convert string device name to hexadecimal before sending.

Parameters **`name`** – the desired name for conerting.

Returns Hexadecimal represntation of the name argument.

Raises **`aioswitcher.erros.EncodingError`** – when failed hexlify.

Note: This is a private function containing blocking code. Please consider using `convert_string_to_device_name` (without the _), to schedule as a task in the event loop.

`aioswitcher.tools.`**`_convert_timedelta_to_auto_off`**(*full_time: datetime.timedelta*) → str

Convert timedelta object for auto-shutdown to hexadecimal.

Parameters **`full_time`** – timedelta object represnting the auto-shutdown time.

Returns Hexadecimal represntation of the full_time argument.

Raises **`aioswitcher.erros.EncodingError`** – when failed hexlify.

Note: This is a private function containing blocking code. Please consider using `convert_timedelta_to_auto_off` (without the _), to schedule as a task in the event loop.

`aioswitcher.tools.`**`_crc_sign_full_packet_com_key`**(*data: str*) → str

Calculate the crc for packets before send.

Parameters **`data`** – packet data to sign.

Returns The calculated and signed packet data.

Raises **`aioswitcher.erros.EncodingError`** – when failed to sign the packet.

Note: This is a private function containing blocking code. Please consider using `crc_sign_full_packet_com_key` (without the _), to schedule as a task in the event loop.

`aioswitcher.tools.`**`_create_weekdays_value`**(*requested_days: List[int]*) → str

Create hex value from list of requested days for schedule updating.

Parameters **data** – list of integers represnting the requested days. check aioswitcher.consts for the correct values.

Returns Hexadecimal representation of the days list.

Raises **aioswitcher.erros.EncodingError** – when failed to convert the argument.

Note: This is a private function containing blocking code. Please consider using create_weekdays_value (without the _), to schedule as a task in the event loop.

---

aioswitcher.tools.**_get_days_list_from_bytes**(*data: int*) → List[str]
Extract week days from shcedule bytes data.

Parameters **data** – bytes representing the days list.

Returns List of string represntation the week days included in the list. See aioswitcher.consts for days literals.

Raises **aioswitcher.erros.DecodingError** – when failed to analyze the argument.

Note: This is a private function containing blocking code. Please consider using get_days_list_from_bytes (without the _), to schedule as a task in the event loop.

---

aioswitcher.tools.**_get_time_from_bytes**(*data: bytes*) → str
Extract start/end time from schedule bytes.

Parameters **data** – bytes representing the start or the end time for the schedule.

Returns %M format. e.g. "20:30".

Return type Time string in %H

Raises **aioswitcher.erros.DecodingError** – when failed to analyze the argument.

Note: This is a private function containing blocking code. Please consider using get_time_from_bytes (without the _), to schedule as a task in the event loop.

---

aioswitcher.tools.**_get_timestamp**() → str
Generate hexadecimal represntation of the current timestamp.

Returns Hexadecimal represntation of the current unix time retrieved by time.time.

Raises **aioswitcher.erros.DecodingError** – when failed to analyze the timestamp.

Note: This is a private function containing blocking code. Please consider using get_timestamp (without the _), to schedule as a task in the event loop.

---

aioswitcher.tools.**_timedelta_str_to_schedule_time**(*time_value: str*) → str
Convert time string to schedule start/end time to hexadecimale.

Parameters **data** – time to convert. e.g. "21:00".

Returns Hexadecimal representation of time_value argument.

Raises **aioswitcher.erros.EncodingError** – when failed to convert the argument.

Note: This is a private function containing blocking code. Please consider using `timedelta_str_to_schedule_time` (without the _), to schedule as a task in the event loop.

async aioswitcher.tools.**convert_minutes_to_timer**(*loop: asyncio.events.AbstractEventLoop*, *minutes: str*) → str

Asynchronous wrapper for _convert_minutes_to_timer.

Use as async wrapper for calling _convert_minutes_to_timer, converting on-timer minutes to hexadecimal.

Parameters **minutes** – on-timer minutes to convert.

Returns Hexadecimal represntation of the mintues argument.

async aioswitcher.tools.**convert_seconds_to_iso_time**(*loop: asyncio.events.AbstractEventLoop*, *all_seconds: int*) → str

Asynchronous wrapper for _convert_seconds_to_iso_time.

Use as async wrapper for calling _convert_seconds_to_iso_time, calculating the next runtime of the schedule.

Parameters

- **loop** – the event loop to execute the function in.
- **all_seconds** – the total number of seconds to convert.

Returns %M:%S format. e.g. "02:24:37".

Return type A string represnting the converted iso time in %H

async aioswitcher.tools.**convert_string_to_device_name**(*loop: asyncio.events.AbstractEventLoop*, *name: str*) → str

Asynchronous wrapper for _convert_string_to_device_name.

Use as async wrapper for calling _convert_string_to_device_name, converting name to hexadecimal.

Parameters **name** – the desired name for conerting.

Returns Hexadecimal represntation of the name argument.

async aioswitcher.tools.**convert_timedelta_to_auto_off**(*loop: asyncio.events.AbstractEventLoop*, *full_time: datetime.timedelta*) → str

Asynchronous wrapper for _convert_timedelta_to_auto_off.

Use as async wrapper for calling _convert_timedelta_to_auto_off, converting timedelta auto-shutdown configuration to hexadecimal.

Parameters **full_time** – timedelta object represnting the auto-shutdown time.

Returns Hexadecimal represntation of the full_time argument.

async aioswitcher.tools.**crc_sign_full_packet_com_key**(*loop: asyncio.events.AbstractEventLoop*, *data: str*) → str

Asynchronous wrapper for _crc_sign_full_packet_com_key.

Use as async wrapper for calling _crc_sign_full_packet_com_key, performing crc sign to the packet data.

Parameters **data** – packet data to sign.

Returns The calculated and signed packet data.

async aioswitcher.tools.**create_weekdays_value**(*loop: asyncio.events.AbstractEventLoop*, *requested_days: List[int]*) → str

Asynchronous wrapper for _create_weekdays_value.

Use as async wrapper for calling _create_weekdays_value, creating hex value from list of requested days for schedule updating.

Parameters **data** – list of integers represnting the requested days. check aioswitcher.consts for the correct values.

Returns Hexadecimal representation of the days list.

async aioswitcher.tools.**get_days_list_from_bytes**(*loop: asyncio.events.AbstractEventLoop*, *data: int*) → List[str]

Asynchronous wrapper for _get_days_list_from_bytes.

Use as async wrapper for calling _get_days_list_from_bytes, extracting week days from shcedule bytes data.

Parameters **data** – bytes representing the days list.

Returns List of string represntation the week days included in the list. See aioswitcher.consts for days literals.

async aioswitcher.tools.**get_time_from_bytes**(*loop: asyncio.events.AbstractEventLoop*, *data: bytes*) → str

Asynchronous wrapper for _get_time_from_bytes.

Use as async wrapper for calling _get_time_from_bytes, extracting start/end time from schedule bytes data.

Parameters **data** – bytes representing the start or the end time for the schedule.

Returns %M format. e.g. “20:30”.

Return type Time string in %H

async aioswitcher.tools.**get_timestamp**(*loop: asyncio.events.AbstractEventLoop*) → str

Asynchronous wrapper for _get_timestamp.

Use as async wrapper for calling _get_timestamp, creating hexadecimal represntation of the current timestamp.

Returns Hexadecimal represntation of the current unix time retrieved by time.time.

async aioswitcher.tools.**timedelta_str_to_schedule_time**(*loop: asyncio.events.AbstractEventLoop*, *time_value: str*) → str

Asynchronous wrapper for _timedelta_str_to_schedule_time.

Use as async wrapper for calling _timedelta_str_to_schedule_time, converting time string to schedule start/end time to hexadecimale.

Parameters **data** – time to convert. e.g. “21:00”.

Returns Hexadecimal representation of time_value argument.

PYTHON MODULE INDEX

a

INDEX

Symbols

A

C

D

E

F

G

H

I

T

U

V

W