

---

# **aioswitcher**

*Release 1.2.0*

**Tomer Figenblat**

**May 17, 2020**



# CONTENTS

<b>1</b>	<b>Install</b>	<b>1</b>
<b>2</b>	<b>Prerequisites</b>	<b>3</b>
<b>3</b>	<b>Usage</b>	<b>5</b>
<b>4</b>	<b>License</b>	<b>17</b>
<b>5</b>	<b>Credits</b>	<b>19</b>
<b>6</b>	<b>Projects using this module</b>	<b>21</b>
<b>7</b>	<b>Code Documentation</b>	<b>23</b>
	<b>Python Module Index</b>	<b>45</b>
	<b>Index</b>	<b>47</b>



## 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.
-



## 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





## 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

- *UDP Bridge*
  - *Example of UDP Bridge usage*
- *TCP Socket API*
  - *Example of TCP Socket API usage*
- *Objects and Properties*
  - *SwitcherV2Device*
  - *SwitcherV2Schedule*
- *API Response Messages*
  - *SwitcherV2BaseResponseMSG*
  - *SwitcherV2StateResponseMSG*
  - *SwitcherV2ControlResponseMSG*
  - *SwitcherV2SetAutoOffResponseMSG*
  - *SwitcherV2UpdateNameResponseMSG*
  - *SwitcherV2GetScheduleResponseMSG*
  - *SwitcherV2DisableEnableScheduleResponseMSG*
  - *SwitcherV2DeleteScheduleResponseMSG*
  - *SwitcherV2CreateScheduleResponseMSG*

## 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
<b>device_id</b>	str	Return the device id.	ab1c2d	
<b>ip_addr</b>	str	Return the ip address.	192.168.100.157	wait- ing_for_data
<b>mac_addr</b>	str	Return the mac address.	A1:B2:C3:45:67:D8	wait- ing_for_data
<b>name</b>	str	Return the device name.	device name	wait- ing_for_data
<b>state</b>	str	Return the device state.	on, off	
<b>remaining_time</b>	str	Return the auto-off configuration value.	%H:%M:%S	wait- ing_for_data
<b>auto_off_set</b>	str	Return the time left to auto-off.	%H:%M:%S	wait- ing_for_data
<b>power_consumption</b>	int	Return the power consumption in watts.	2780	0
<b>electric_current</b>	float	Return the power consumption in amps.	12.8	0.0
<b>phone_id</b>	str	Return the the phone id.	1234	
<b>last_data_update</b>	datetime	Return the timestamp of the last update.	%Y-%m- %dTH:%M:%S.%F	
<b>last_state_change</b>	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
<b>schedule_id</b>	str	Return the schedule id.	0-7	
<b>enabled</b>	bool	Return true if enabled. Has a setter manipulating the schedule status.	True, False	False
<b>recurring</b>	bool	Return true if recurring.	True, False	False
<b>days</b>	List[str]	Return the weekdays of the schedule.	<ul style="list-style-type: none"> <li>• Sunday</li> <li>• Monday</li> <li>• Tuesday</li> <li>• Wednesday</li> <li>• Thursday</li> <li>• Friday</li> <li>• Saturday</li> <li>• <b>Every day</b></li> </ul>	
<b>start_time</b>	str	Return the start time of the schedule.	%H:%M	waiting_for_data
<b>end_time</b>	str	Return the end time of the schedule.	%H:%M	waiting_for_data
<b>duration</b>	str	Return the duration time of the schedule.	0:30:00	waiting_for_data
<b>schedule_data</b>	str	Return the schedule data for managing the schedule. has a setter manipulating the schedule data.	Any	waiting_for_data
<b>init_future</b>	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.messages.ResponseMessageType 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
<code>unparsed_response</code>	<code>bytes</code>	Return the unparsed response message.
<code>successful</code>	<code>bool</code>	Return the status of the message.
<code>msg_type</code>	<code>ResponseMessageType</code>	Return the response message type.

### 3.4.2 SwitcherV2StateResponseMSG

**Extends** `SwitcherV2BaseResponseMSG`

**Response Type** `ResponseMessageType.STATE`

Property	Type	Description
<code>state</code>	<code>str</code>	Return the state. Possible values are: <ul style="list-style-type: none"> <li>• <code>aioswitcher.consts.STATE_ON</code></li> <li>• <code>aioswitcher.consts.STATE_OFF</code></li> </ul>
<code>time_left</code>	<code>str</code>	Return the time left to auto-off.
<code>time_on</code>	<code>str</code>	Return the time in “on” state. Relevant only if the current state is “on”.
<code>auto_off</code>	<code>str</code>	Return the auto-off configuration value.
<code>power</code>	<code>Optional[int]</code>	Return the current power consumption in watts.
<code>current</code>	<code>Optional[float]</code>	Return the power consumption in amps.
<code>init_future</code>	<code>asyncio.Future</code>	Return the future of the initialization. As the initialization of this message requires some asynchronous actions, please use <code>init_future.result()</code> 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
<b>found_schedules</b>	<code>bool</code>	Return true if found schedules in the response.
<b>get_schedules</b>	<code>List [SwitcherV2Schedule]</code>	Return a list of <i>SwitcherV2Schedule</i> .

### 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`.



**LICENSE**

MIT License

Copyright © 2019 Tomer Figenblat

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



## 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!





## 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).

---



## 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*)

Representation 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 designated 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 containing 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 containing 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 containing the configuration value for auto-shutdown. Accepts anything 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  $2 \leq x \leq 32$ .

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

**async get\_schedules** ()

Use for retrieval 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  
    Bases: enum.Enum
```

An enumeration.

```
class aioswitcher.api.messages.SwitcherV2BaseResponseMSG (loop:          asyn-  
                                                         cio.events.AbstractEventLoop,  
                                                         response:        bytes,  
                                                         msg_type:        aioswitcher.api.messages.ResponseMessageType)
```

Representation 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:          asyn-  
                                                         cio.events.AbstractEventLoop,  
                                                         response: bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation 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:
                                                    asyn-
                                                    cio.events.AbstractEventLoop,
                                                    response:
                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation 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:
                                                    asyn-
                                                    cio.events.AbstractEventLoop,
                                                    response:
                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation 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:
                                                    asyn-
                                                    cio.events.AbstractEvent
                                                    re-
                                                    sponse:
                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation 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*

representation 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.
  - `schedule_details` 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:          asyn-
                                                           cio.events.AbstractEventLoop,
                                                           response: bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation 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:          asyn-
                                                                    cio.events.AbstractEventLoop,
                                                                    response:
                                                                    bytes)
```

Bases: *aioswitcher.api.messages.SwitcherV2BaseResponseMSG*

Representation 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*

Representation 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*

Representation 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)
```

Representation 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 listener 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])
```

Representation 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 consumption in 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 consumption in 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)
```

Representation 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 consumption in watts.
- **electric\_current** – the current power consumption 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 consumption in watts.
- **electric\_current** – the current power consumption in amps.
- **last\_state\_change** – datetime of the last state change.

**as\_dict ()**

Return as dict.

**Returns** A dictionary representation 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`

Representation 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 received.

**error\_received** (*exc*)

Call on exception received.

**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 received.

**error\_received** (*exc: Optional[Exception]*) → None

Call on exception received.

**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: async-  
                                               cio.events.AbstractEventLoop, idx:  
                                               int, schedule_details: List[bytes])
```

Representation 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 representation 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

`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 tomorrow 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** `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 tomorrow 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 representation of the minutes argument.

**Raises** `aioswitcher.errors.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 representing the converted iso time in `%H`

**Raises** `aioswitcher.errors.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 representation of the name argument.

**Raises** `aioswitcher.errors.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 representing the auto-shutdown time.

**Returns** Hexadecimal representation of the `full_time` argument.

**Raises** `aioswitcher.errors.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.errors.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 representing the requested days. check `aioswitcher.consts` for the correct values.

**Returns** Hexadecimal representation of the days list.

**Raises** `aioswitcher.errors.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 representation the week days included in the list. See `aioswitcher.consts` for days literals.

**Raises** `aioswitcher.errors.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.errors.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 representation of the current timestamp.

**Returns** Hexadecimal representation of the current unix time retrieved by `time.time`.

**Raises** `aioswitcher.errors.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.errors.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:* *async-*  
*cio.events.AbstractEventLoop,*  
*minutes: str*)  $\rightarrow$  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 representation of the minutes argument.

**async** `aioswitcher.tools.convert_seconds_to_iso_time` (*loop:* *async-*  
*cio.events.AbstractEventLoop,*  
*all\_seconds: int*)  $\rightarrow$  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 representing the converted iso time in `%H`

**async** `aioswitcher.tools.convert_string_to_device_name` (*loop:* *async-*  
*cio.events.AbstractEventLoop,*  
*name: str*)  $\rightarrow$  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 converting.

**Returns** Hexadecimal representation of the name argument.

**async** `aioswitcher.tools.convert_timedelta_to_auto_off` (*loop:* *async-*  
*cio.events.AbstractEventLoop,*  
*full\_time:* *date-*  
*time.timedelta*)  $\rightarrow$  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 representing the auto-shutdown time.

**Returns** Hexadecimal representation of the `full_time` argument.

**async** `aioswitcher.tools.crc_sign_full_packet_com_key` (*loop:* *async-*  
*cio.events.AbstractEventLoop,*  
*data: str*)  $\rightarrow$  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 representing 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 schedule bytes data.

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

**Returns** List of string representation 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 representation of the current timestamp.

**Returns** Hexadecimal representation 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 hexadecimal.

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

**Returns** Hexadecimal representation of `time_value` argument.





## PYTHON MODULE INDEX

### a

- [aioswitcher](#), 23
- [aioswitcher.api](#), 23
  - [aioswitcher.api.messages](#), 26
  - [aioswitcher.api.packets](#), 31
- [aioswitcher.bridge](#), 31
  - [aioswitcher.bridge.messages](#), 32
- [aioswitcher.consts](#), 34
- [aioswitcher.devices](#), 34
- [aioswitcher.errors](#), 36
- [aioswitcher.protocols](#), 36
- [aioswitcher.schedules](#), 38
- [aioswitcher.tools](#), 39



## Symbols

- `__aenter__()` (*aioswitcher.api.SwitcherV2Api method*), 23
  - `__aenter__()` (*aioswitcher.bridge.SwitcherV2Bridge method*), 32
  - `__aexit__()` (*aioswitcher.api.SwitcherV2Api method*), 23
  - `__aexit__()` (*aioswitcher.bridge.SwitcherV2Bridge method*), 32
  - `__await__()` (*aioswitcher.api.SwitcherV2Api method*), 23
  - `__await__()` (*aioswitcher.bridge.SwitcherV2Bridge method*), 32
  - `_calc_next_run_for_schedule()` (*aioswitcher.schedules method*), 39
  - `_convert_minutes_to_timer()` (in module *aioswitcher.tools*), 39
  - `_convert_seconds_to_iso_time()` (in module *aioswitcher.tools*), 40
  - `_convert_string_to_device_name()` (in module *aioswitcher.tools*), 40
  - `_convert_timedelta_to_auto_off()` (in module *aioswitcher.tools*), 40
  - `_crc_sign_full_packet_com_key()` (in module *aioswitcher.tools*), 40
  - `_create_weekdays_value()` (in module *aioswitcher.tools*), 40
  - `_full_get_state()` (*aioswitcher.api.SwitcherV2Api method*), 24
  - `_full_login()` (*aioswitcher.api.SwitcherV2Api method*), 24
  - `_get_days_list_from_bytes()` (in module *aioswitcher.tools*), 41
  - `_get_time_from_bytes()` (in module *aioswitcher.tools*), 41
  - `_get_timestamp()` (in module *aioswitcher.tools*), 41
  - `_timedelta_str_to_schedule_time()` (in module *aioswitcher.tools*), 41
  - `aioswitcher`
    - module, 23
  - `aioswitcher.api`
    - module, 23
  - `aioswitcher.api.messages`
    - module, 26
  - `aioswitcher.api.packets`
    - module, 31
  - `aioswitcher.bridge`
    - module, 31
  - `aioswitcher.bridge.messages`
    - module, 32
  - `aioswitcher.consts`
    - module, 34
  - `aioswitcher.devices`
    - module, 34
  - `aioswitcher.errors`
    - module, 36
  - `aioswitcher.protocols`
    - module, 36
  - `aioswitcher.schedules`
    - module, 38
  - `aioswitcher.tools`
    - module, 39
  - `as_dict()` (*aioswitcher.devices.SwitcherV2Device method*), 34
  - `as_dict()` (*aioswitcher.schedules.SwitcherV2Schedule method*), 38
  - `auto_off()` (*aioswitcher.api.messages.SwitcherV2StateResponseMSG property*), 30
  - `auto_off_set()` (*aioswitcher.bridge.messages.SwitcherV2BroadcastM property*), 33
  - `auto_off_set()` (*aioswitcher.devices.SwitcherV2Device property*), 35
- ## C
- `calc_next_run_for_schedule()` (*aioswitcher.schedules method*), 39
  - CalculationError, 36
  - `close_transport()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory method*), 37

## A

aioswitcher

`connect()` (*aioswitcher.api.SwitcherV2Api* method), 23  
`connected()` (*aioswitcher.api.SwitcherV2Api* property), 25  
`connection_lost()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 37  
`connection_made()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 37  
`convert_minutes_to_timer()` (in module *aioswitcher.tools*), 42  
`convert_seconds_to_iso_time()` (in module *aioswitcher.tools*), 42  
`convert_string_to_device_name()` (in module *aioswitcher.tools*), 42  
`convert_timedelta_to_auto_off()` (in module *aioswitcher.tools*), 42  
`crc_sign_full_packet_com_key()` (in module *aioswitcher.tools*), 42  
`create_schedule()` (*aioswitcher.api.SwitcherV2Api* method), 25  
`create_weekdays_value()` (in module *aioswitcher.tools*), 43  
`current()` (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 30  
`current()` (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 33  
**D**  
`datagram_received()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 37  
`days()` (*aioswitcher.schedules.SwitcherV2Schedule* property), 38  
`DecodingError`, 36  
`delete_schedule()` (*aioswitcher.api.SwitcherV2Api* method), 25  
`device_id()` (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 33  
`device_id()` (*aioswitcher.devices.SwitcherV2Device* property), 35  
`device_password()` (*aioswitcher.devices.SwitcherV2Device* property), 35  
`device_state()` (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 33  
`disable_enable_schedule()` (*aioswitcher.api.SwitcherV2Api* method), 25  
`disconnect()` (*aioswitcher.api.SwitcherV2Api* method), 24  
`duration()` (*aioswitcher.schedules.SwitcherV2Schedule* property), 38  
**E**  
`electric_current()` (*aioswitcher.devices.SwitcherV2Device* property), 35  
`enabled()` (*aioswitcher.schedules.SwitcherV2Schedule* property), 38  
`EncodingError`, 36  
`end_time()` (*aioswitcher.schedules.SwitcherV2Schedule* property), 38  
`error_received()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 37  
**F**  
`factory_future()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* property), 37  
`found_schedules()` (*aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG* property), 28  
**G**  
`get_days_list_from_bytes()` (in module *aioswitcher.tools*), 43  
`get_device_from_message()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 37  
`get_schedules()` (*aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG* property), 28  
`get_schedules()` (*aioswitcher.api.SwitcherV2Api* method), 25  
`get_state()` (*aioswitcher.api.SwitcherV2Api* method), 24  
`get_time_from_bytes()` (in module *aioswitcher.tools*), 43  
`get_timestamp()` (in module *aioswitcher.tools*), 43  
**H**  
`handle_incoming_messages()` (*aioswitcher.protocols.SwitcherV2UdpProtocolFactory* method), 37  
**I**  
`init_future()` (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* property), 30  
`init_future()` (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* property), 33  
`init_future()` (*aioswitcher.schedules.SwitcherV2Schedule* property), 39  
`initialize()` (*aioswitcher.api.messages.SwitcherV2StateResponseMSG* method), 30  
`initialize()` (*aioswitcher.bridge.messages.SwitcherV2BroadcastMSG* method), 32

initialize() (aioswitcher.schedules.SwitcherV2Schedule  
 method), 38  
 ip\_addr() (aioswitcher.devices.SwitcherV2Device  
 property), 35  
 ip\_address() (aioswitcher.bridge.messages.SwitcherV2BroadcastMSG  
 property), 33

## L

last\_data\_update() (aioswitcher.devices.SwitcherV2Device prop-  
 erty), 35  
 last\_state\_change() (aioswitcher.devices.SwitcherV2Device prop-  
 erty), 35  
 login() (aioswitcher.api.SwitcherV2Api method), 24

## M

mac\_addr() (aioswitcher.devices.SwitcherV2Device  
 property), 35  
 mac\_address() (aioswitcher.bridge.messages.SwitcherV2BroadcastMSG  
 property), 33  
 module  
   aioswitcher, 23  
   aioswitcher.api, 23  
   aioswitcher.api.messages, 26  
   aioswitcher.api.packets, 31  
   aioswitcher.bridge, 31  
   aioswitcher.bridge.messages, 32  
   aioswitcher.consts, 34  
   aioswitcher.devices, 34  
   aioswitcher.errors, 36  
   aioswitcher.protocols, 36  
   aioswitcher.schedules, 38  
   aioswitcher.tools, 39  
 msg\_type() (aioswitcher.api.messages.SwitcherV2BaseResponseMSG  
 property), 26  
 msg\_type() (aioswitcher.api.messages.SwitcherV2ControlResponseMSG  
 property), 26  
 msg\_type() (aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG  
 property), 27  
 msg\_type() (aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG  
 property), 27  
 msg\_type() (aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG  
 property), 28  
 msg\_type() (aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG  
 property), 28  
 msg\_type() (aioswitcher.api.messages.SwitcherV2LoginResponseMSG  
 property), 29  
 msg\_type() (aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG  
 property), 29  
 msg\_type() (aioswitcher.api.messages.SwitcherV2StateResponseMSG  
 property), 30  
 msg\_type() (aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG  
 property), 31

## N

name() (aioswitcher.bridge.messages.SwitcherV2BroadcastMSG  
 property), 33  
 name() (aioswitcher.devices.SwitcherV2Device prop-  
 erty), 35

## P

pause\_writing() (aioswitcher.protocols.SwitcherV2UdpProtocolFacto-  
 ry), 37  
 phone\_id() (aioswitcher.devices.SwitcherV2Device  
 property), 35  
 power() (aioswitcher.api.messages.SwitcherV2StateResponseMSG  
 property), 30  
 power() (aioswitcher.bridge.messages.SwitcherV2BroadcastMSG  
 property), 33  
 power\_consumption() (aioswitcher.devices.SwitcherV2Device prop-  
 erty), 35

## Q

queue() (aioswitcher.bridge.SwitcherV2Bridge prop-  
 erty), 32

## R

recurring() (aioswitcher.schedules.SwitcherV2Schedule  
 property), 39  
 remaining\_time() (aioswitcher.devices.SwitcherV2Device  
 property), 35  
 remaining\_time\_to\_off() (aioswitcher.bridge.messages.SwitcherV2BroadcastMSG  
 property), 33  
 ResponseMessageType (class in  
 aioswitcher.api.messages), 26  
 resume\_writing() (aioswitcher.protocols.SwitcherV2UdpProtocolFacto-  
 ry), 38  
 running() (aioswitcher.bridge.SwitcherV2Bridge  
 property), 32

## S

schedule\_data() (aioswitcher.schedules.SwitcherV2Schedule  
 property), 39  
 schedule\_id() (aioswitcher.schedules.SwitcherV2Schedule  
 property), 39  
 session\_id() (aioswitcher.api.messages.SwitcherV2LoginResponseMSG  
 property), 29  
 set\_auto\_shutdown() (aioswitcher.api.SwitcherV2Api method),  
 24  
 set\_device\_name() (aioswitcher.api.SwitcherV2Api method),  
 25  
 start() (aioswitcher.bridge.SwitcherV2Bridge  
 method), 32

start\_time() (aioswitcher.schedules.SwitcherV2Schedule property), 39  
 state() (aioswitcher.api.messages.SwitcherV2StateResponseMSG property), 31  
 state() (aioswitcher.devices.SwitcherV2Device property), 35  
 stop() (aioswitcher.bridge.SwitcherV2Bridge method), 32  
 successful() (aioswitcher.api.messages.SwitcherV2BaseResponseMSG property), 26  
 successful() (aioswitcher.api.messages.SwitcherV2ControlResponseMSG property), 26  
 successful() (aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG property), 27  
 successful() (aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG property), 27  
 successful() (aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG property), 28  
 successful() (aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG property), 28  
 successful() (aioswitcher.api.messages.SwitcherV2LoginResponseMSG property), 29  
 successful() (aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG property), 29  
 successful() (aioswitcher.api.messages.SwitcherV2StateResponseMSG property), 30  
 successful() (aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG property), 31  
 SwitcherV2Api (class in aioswitcher.api), 23  
 SwitcherV2BaseResponseMSG (class in aioswitcher.api.messages), 26  
 SwitcherV2Bridge (class in aioswitcher.bridge), 31  
 SwitcherV2BroadcastMSG (class in aioswitcher.bridge.messages), 32  
 SwitcherV2ControlResponseMSG (class in aioswitcher.api.messages), 26  
 SwitcherV2CreateScheduleResponseMSG (class in aioswitcher.api.messages), 27  
 SwitcherV2DeleteScheduleResponseMSG (class in aioswitcher.api.messages), 27  
 SwitcherV2Device (class in aioswitcher.devices), 34  
 SwitcherV2DisableEnableScheduleResponseMSG (class in aioswitcher.api.messages), 27  
 SwitcherV2GetScheduleResponseMSG (class in aioswitcher.api.messages), 28  
 SwitcherV2LoginResponseMSG (class in aioswitcher.api.messages), 29  
 SwitcherV2Schedule (class in aioswitcher.schedules), 38  
 SwitcherV2SetAutoOffResponseMSG (class in aioswitcher.api.messages), 29  
 SwitcherV2StateResponseMSG (class in aioswitcher.api.messages), 29  
 SwitcherV2UdpProtocolFactory (class in aioswitcher.protocols), 36  
 SwitcherV2UpdateNameResponseMSG (class in aioswitcher.api.messages), 31  
 T  
 time\_left() (aioswitcher.api.messages.SwitcherV2StateResponseMSG property), 30  
 time\_on() (aioswitcher.api.messages.SwitcherV2StateResponseMSG property), 30  
 timedelta\_str\_to\_schedule\_time() (in module aioswitcher.tools), 43  
 U  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2BaseResponseMSG property), 26  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2ControlResponseMSG property), 26  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2CreateScheduleResponseMSG property), 27  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2DeleteScheduleResponseMSG property), 27  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2DisableEnableScheduleResponseMSG property), 28  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG property), 28  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2LoginResponseMSG property), 29  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG property), 29  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2StateResponseMSG property), 30  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG property), 31  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2GetScheduleResponseMSG property), 28  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2LoginResponseMSG property), 29  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2SetAutoOffResponseMSG property), 29  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2StateResponseMSG property), 31  
 unparsed\_response() (aioswitcher.api.messages.SwitcherV2UpdateNameResponseMSG property), 31  
 update\_device\_data() (aioswitcher.devices.SwitcherV2Device method), 34  
 V  
 verified() (aioswitcher.bridge.messages.SwitcherV2BroadcastMSG property), 33  
 W  
 with\_traceback() (aioswitcher.errors.CalculationError method), 36

`with_traceback()` (*aioswitcher.errors.DecodingError*  
*method*), 36

`with_traceback()` (*aioswitcher.errors.EncodingError*  
*method*), 36