# AlexaPy

*Release 1.25.3*

**unknown**

**2022-05-29**

# CONTENTS:

CHAPTER

ONE

# ALEXAPY

Python Package for controlling Alexa devices (echo dot, etc) programmatically. This was originally designed for alexa_media_player a custom_component for Home Assistant.

**NOTE:** Alexa has no official API; therefore, this library may stop working at any time without warning.

CHAPTER

TWO

# CREDITS

Originally inspired by this blog (GitHub). Additional scaffolding from simplisafe-python

CHAPTER

THREE

# CONTRIBUTING

1. Check for open features/bugs or initiate a discussion on one.
2. Fork the repository.
3. Install the dev environment: `make init`.
4. Enter the virtual environment: `pipenv shell`
5. Code your new feature or bug fix.
6. Write a test that covers your new functionality.
7. Update `README.md` with any new documentation.
8. Run tests and ensure 100% code coverage for your contribution: `make coverage`
9. Ensure you have no linting errors: `make lint`
10. Ensure you have typed your code correctly: `make typing`
11. Add yourself to `AUTHORS.md`.
12. Submit a pull request!

CHAPTER

FOUR

# LICENSE

Apache-2.0. By providing a contribution, you agree the contribution is licensed under Apache-2.0.

CHAPTER

FIVE

# API REFERENCE

See the docs .

CHAPTER

SIX

# API REFERENCE

## 6.1 alexapy

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy



### 6.1.1 Submodules

#### alexapy.alexaapi

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

API access.

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy

#### alexapy.alexalogin

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

Login class.

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy

### alexapy.alexaproxy

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

This provides a login by proxy method. Built on https://github.com/alandtse/auth_capture_proxy

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy

### alexapy.alexawebsocket

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

Websocket library.

This library is based on MIT code from https://github.com/Apollon77/alexa-remote.

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy

### alexapy.const

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

Constants.

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy

### alexapy.errors

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

Package errors.

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy

### alexapy.helpers

Python Package for controlling Alexa devices (echo dot, etc) programmatically.

SPDX-License-Identifier: Apache-2.0

Helpers.

For more details about this api, please refer to the documentation at https://gitlab.com/keatontaylor/alexapy

### 6.1.2 Functions

- *hide_email()*: Obfuscate email.
- *hide_serial()*: Obfuscate serial.
- *obfuscate()*: Obfuscate email, password, and other known sensitive keys.

alexapy.**hide_email**(*email: str*) → str

Obfuscate email.

alexapy.**hide_serial**(*item: Optional[Union[dict, str, list]]*) → Union[dict, str, list]

Obfuscate serial.

alexapy.**obfuscate**(*item*)

Obfuscate email, password, and other known sensitive keys.

### 6.1.3 Classes

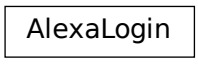
- *AlexaLogin*: Class to handle login connection to Alexa. This class will not reconnect.
- *AlexaAPI*: Class for accessing a specific Alexa device using rest API.
- *AlexaProxy*: Class to handle proxy login connections to Alexa.
- *WebsocketEchoClient*: WebSocket Client Class for Echo Devices.

**class** alexapy.**AlexaLogin**(*url: str*, *email: str*, *password: str*, *outputpath: Callable[[str], str]*, *debug: bool = False*, *otp_secret: str = ''*, *oauth: Optional[Dict[Any, Any]] = None*, *uuid: Optional[str] = None*, *oauth_login: bool = True*)

Class to handle login connection to Alexa. This class will not reconnect.

Args: url (string): Localized Amazon domain (e.g., amazon.com) email (string): Amazon login account password (string): Password for Amazon login account outputpath (function): Local path with write access for storing files debug (boolean): Enable additional debugging including debug file creation otp_secret (string): TOTP Secret key for automatic 2FA filling uuid: (string): Unique 32 char hex to serve as app serial number for registration

#### Inheritance




**async check_domain**() → bool

Check whether logged into appropriate login domain.

**Returns** bool: True if in correct domain

**async close**() → None

Close connection for login.

property close_requested: bool
Return whether this Login has been asked to close.

property customer_id: Optional[str]
Return customer_id for this Login.

property email: str
Return email or mobile account for this Login.

async exchange_token_for_cookies() → bool
Generate new session cookies using refresh token.

**Returns** bool: True if succesful

async finalize_login() → None
Perform final steps after successful login.

async get_csrf() → bool
Generate csrf if missing.

**Returns** bool: True if csrf is found

classmethod get_inputs(*soup: bs4.BeautifulSoup*, *searchfield=None*) → Dict[str, str]
Parse soup for form with searchfield.

async get_tokens() → bool
Get access and refresh tokens after registering device using cookies.

**Returns** True if successful.

**Return type** bool

get_totp_token() → str
Generate Timed based OTP token.

**Returns** Text: OTP for current time.

property lastreq: Optional[alexapy.aiohttp.client_reqrep.ClientResponse]
Return last response for last request for this Login.

property links: str
Return string list of links from last page for this Login.

async load_cookie(*cookies_txt: str = ''*) → Optional[Dict[str, str]]
Load cookie from disk.

async login(*cookies: Optional[Dict[str, str]] = None*, *data: Optional[Dict[str, Optional[str]]] = None*) → None
Login to Amazon.

property password: str
Return password for this Login.

async refresh_access_token() → bool
Refresh access token and expires in using refresh token.

**Returns** bool: Return true if successful.

async reset() → None
Remove data related to existing login.

**async save_cookiefile**() → None

Save login session cookies to file.

**property session: Optional[alexapy.aiohttp.client.ClientSession]**

Return session for this Login.

**set_totp**(*otp_secret: str*) → Optional[pyotp.totp.TOTP]

Enable a TOTP generator for the login.

**Args** otp_secret (Text): Secret. If blank, it will remove the TOTP entry.

**Returns** Optional[pyotp.TOTP]: The pyotp TOTP object

**property start_url: yarl.URL**

Return start url for this Login.

**async test_loggedin**(*cookies: Optional[Dict[str, str]] = None*) → bool

Function that will test the connection is logged in.

Tests: - Attempts to get authenticaton and compares to expected login email Returns false if unsuccesful getting json or the emails don't match Returns false if no csrf found; necessary to issue commands

**property url: str**

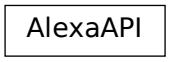
Return url for this Login.

**class** alexapy.**AlexaAPI**(*device*, *login:* alexapy.alexalogin.AlexaLogin)

Class for accessing a specific Alexa device using rest API.

Args: device (AlexaClient): Instance of an AlexaClient to access login (AlexaLogin): Successfully logged in AlexaLogin

### Inheritance




**async static clear_history**(*login:* alexapy.alexalogin.AlexaLogin, *items: int = 50*) → bool

Clear entries in history.

**async disconnect_bluetooth**() → None

Disconnect all bluetooth devices.

**async static force_logout**() → None

Force logout.

**Raises** AlexapyLoginError: Raise AlexapyLoginError

**async forward**() → None

Fastforward.

**async static get_activities**(*login:* alexapy.alexalogin.AlexaLogin, *items: int = 10*) → Optional[Dict[str, Any]]

Get activities json.

**async static get_authentication**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Get authentication json.

**async static get_automations**(*login:* alexapy.alexalogin.AlexaLogin, *items: int = 1000*) → Optional[Dict[str, Any]]

Identify all Alexa automations.

**async static get_bluetooth**(*login*) → Optional[Dict[str, Any]]

Get paired bluetooth devices.

**async static get_device_preferences**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Identify all Alexa device preferences.

**async static get_devices**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Identify all Alexa devices.

**async static get_dnd_state**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Get Alexa DND states.

Args: login (AlexaLogin): Successfully logged in AlexaLogin

Returns json

**async static get_entity_state**(*login:* alexapy.alexalogin.AlexaLogin, *entity_ids: Optional[List[str]] = None*, *appliance_ids: Optional[List[str]] = None*) → Optional[Dict[str, Any]]

Get the current state of multiple appliances.

Note that this can take both entity_ids and appliance_ids. If you have both pieces of data available, prefer the entity id. A single entity might have multiple appliance ids. Its easier to ensure you don't miss data by just providing entity id instead.

Args: login (AlexaLogin): Successfully logged in AlexaLogin entity_ids (List[Text]): The list of entities you want information about. appliance_ids: (List[Text]): The list of appliances you want information about.

Returns json

**async static get_guard_details**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Get Alexa Guard details.

Args: login (AlexaLogin): Successfully logged in AlexaLogin

Returns json

**async static get_guard_state**(*login:* alexapy.alexalogin.AlexaLogin, *entity_id: str*) → Optional[Dict[str, Any]]

Get state of Alexa guard.

Args: login (AlexaLogin): Successfully logged in AlexaLogin entity_id (Text): applianceId of RedRock Panel

Returns json

**async static get_last_device_serial**(*login:* alexapy.alexalogin.AlexaLogin, *items: int = 10*) → Optional[Dict[str, Any]]

Identify the last device's serial number and last summary.

This will search the [last items] activity records and find the latest entry where Echo successfully responded.

**async static get_network_details**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Get the network of devices that Alexa is aware of. This is the same as calling get_guard_details().

Args: login: (AlexaLogin): Successfully logged in AlexaLogin

Returns json

**async static get_notifications**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Get Alexa notifications.

Args: login (AlexaLogin): Successfully logged in AlexaLogin

Returns json

**async get_state**() → Optional[Dict[str, Any]]

Get playing state.

**async next**() → None

Play next.

**async pause**() → None

Pause.

**async static ping**(*login:* alexapy.alexalogin.AlexaLogin) → Optional[Dict[str, Any]]

Ping.

Args: login (AlexaLogin): Successfully logged in AlexaLogin

Returns json

**async play**() → None

Play.

**async play_music**(*provider_id: str*, *search_phrase: str*, *customer_id: Optional[str] = None*, *timer: Optional[int] = None*, *queue_delay: float = 1.5*, *extra: Optional[Dict[Any, Any]] = None*) → None

Play music based on search.

**Parameters**

- **provider_id** (*Text*) – Amazon music provider.
- **search_phrase** (*Text*) – Phrase to be searched for
- **customer_id** (*Optional[Text], optional*) – CustomerId to use for authorization. When none specified this defaults to the logged in user. Used with households where others may have their own music.
- **timer** (*Optional[int]*) – Number of seconds to play before stopping.
- **queue_delay** (*float, optional*) – The number of seconds to wait for commands to queue together. Must be positive. Defaults to 1.5.
- **extra** (*Dict*) – Extra dictionary array; functionality undetermined

**async play_sound**(*sound_string_id: str*, *customer_id: Optional[str] = None*, *queue_delay: float = 1.5*, *extra: Optional[Dict[Any, Any]] = None*) → None

Play Alexa sound.

**async previous**() → None

Play previous.

**process_targets**(*targets: Optional[List[str]] = None*) → List[Dict[str, str]]

Process targets list to generate list of devices.

**Keyword Arguments**

**targets {Optional[List[Text]]} – List of serial numbers** (default: {[]})

**Returns** List[Dict[Text, Text] – List of device dicts

**async repeat**(*setting: bool*) → None

Repeat.

setting (string) : true or false

**async rewind**() → None

Rewind.

**async run_behavior**(*node_data*, *queue_delay: float = 1.5*) → None

Queue node_data for running a behavior in sequence.

Amazon sequences and routines are based on node_data.

**Parameters**

- **node_data** (*dict, list of dicts*) – The node_data to run.
- **queue_delay** (*float, optional*) – The number of seconds to wait for commands to queue together. Defaults to 1.5. Must be positive.

**async run_custom**(*text: str*, *customer_id: Optional[str] = None*, *queue_delay: float = 0*, *extra: Optional[Dict[Any, Any]] = None*) → None

Run Alexa skill.

This allows running exactly what you can say to alexa.

**Parameters**

- **text** (*string*) – The full text you want alexa to execute.
- **customer_id** (*string*) – CustomerId to use for authorization. When none specified this defaults to the logged in user. Used with households where others may have their own music.
- **queue_delay** (*float, optional*) – The number of seconds to wait for commands to queue together. Defaults to 1.5. Must be positive.
- **extra** (*Dict*) – Extra dictionary array; functionality undetermined

**async run_routine**(*utterance: str*, *customer_id: Optional[str] = None*, *queue_delay: float = 1.5*) → None

Run Alexa automation routine.

This allows running of defined Alexa automation routines.

**Parameters**

- **utterance** (*string*) – The Alexa utterance to run the routine.

- **customer_id** (*string*) – CustomerId to use for authorization. When none specified this defaults to the logged in user. Used with households where others may have their own music.
- **queue_delay** (*float, optional*) – The number of seconds to wait for commands to queue together. Defaults to 1.5. Must be positive.

**async run_skill**(*skill_id: str*, *customer_id: Optional[str] = None*, *queue_delay: float = 0*) → None

Run Alexa skill.

This allows running of defined Alexa skill.

**Parameters**

- **skill_id** (*string*) – The full skill id.
- **customer_id** (*string*) – CustomerId to use for authorization. When none specified this defaults to the logged in user. Used with households where others may have their own music.
- **queue_delay** (*float, optional*) – The number of seconds to wait for commands to queue together. Defaults to 1.5. Must be positive.

**async send_announcement**(*message: str*, *method: str = 'all'*, *title: str = 'Announcement'*, *customer_id: Optional[str] = None*, *targets: Optional[List[str]] = None*, *queue_delay: float = 1.5*, *extra: Optional[Dict[Any, Any]] = None*) → None

Send announcment to Alexa devices.

This uses the AlexaAnnouncement and allows visual display on the Show. It will beep prior to speaking.

Args: message (string): The message to speak or display. method (string): speak, show, or all title (string): title to display on Echo show customer_id (string): CustomerId to use for authorization. When none

specified this defaults to the logged in user. Used with households where others may have their own music.

**targets (list(string)): List of serialNumber or accountName to send the** announcement to. Only those in this AlexaAPI account will be searched. If None, announce will be self.

**queue_delay (float, optional): The number of seconds to wait** for commands to queue together. Defaults to 1.5. Must be positive.

extra (Dict): Extra dictionary array; functionality undetermined

**async send_dropin_notification**(*message: str*, *title: str = 'AlexaAPI Dropin Notification'*, *customer_id: Optional[str] = None*, *queue_delay: float = 1.5*, *extra: Optional[Dict[Any, Any]] = None*) → None

Send dropin notification to Alexa app for Alexa device.

Push a message to mobile devices with the Alexa App. This can spawn a notification to drop in on a specific device.

Args: message (string): The message to push to the mobile device. title (string): Title for push notification customer_id (string): CustomerId to use for sending. When none

specified this defaults to the logged in user.

**queue_delay (float, optional): The number of seconds to wait** for commands to queue together. Defaults to 1.5. Must be positive.

extra (Dict): Extra dictionary array; functionality undetermined

**async send_mobilepush**(*message: str*, *title: str = 'AlexaAPI Message'*, *customer_id: Optional[str] = None*, *queue_delay: float = 1.5*, *extra: Optional[Dict[Any, Any]] = None*) → None

Send mobile push to Alexa app.

Push a message to mobile devices with the Alexa App. This probably should be a static method.

Args: message (string): The message to push to the mobile device. title (string): Title for push notification customer_id (string): CustomerId to use for sending. When none

specified this defaults to the logged in user.

**queue_delay (float, optional): The number of seconds to wait** for commands to queue together. Defaults to 1.5. Must be positive.

extra (Dict): Extra dictionary array; functionality undetermined

**async send_sequence**(*sequence: str*, *customer_id: Optional[str] = None*, *queue_delay: float = 1.5*, *extra: Optional[Dict[Any, Any]] = None*, ***kwargs*) → None

Send sequence command.

This allows some programatic control of Echo device using the behaviors API and is the basis of play_music, send_announcement, and send_tts.

Args: sequence (string): The Alexa sequence. Supported list below. customer_id (string): CustomerId to use for authorization. When none

specified this defaults to the logged in user. Used with households where others may have their own music.

**queue_delay (float, optional): The number of seconds to wait** for commands to queue together. Defaults to 1.5. Must be positive.

extra (Dict): Extra dictionary array; functionality undetermined **kwargs : Each named variable must match a recognized Amazon variable

within the operationPayload. Please see examples in play_music, send_announcement, and send_tts. Variables with value None are removed from the operationPayload. Variables prefixed with "**root_**" will be added to the root node instead.

Supported sequences: Alexa.Weather.Play Alexa.Traffic.Play Alexa.FlashBriefing.Play Alexa.GoodMorning.Play Alexa.GoodNight.Play Alexa.SingASong.Play Alexa.TellStory.Play Alexa.FunFact.Play Alexa.Joke.Play Alexa.CleanUp.Play Alexa.Music.PlaySearchPhrase Alexa.Calendar.PlayTomorrow Alexa.Calendar.PlayToday Alexa.Calendar.PlayNext https://github.com/custom-components/alexa_media_player/wiki#sequence-commands-versions–100

**async send_tts**(*message: str*, *customer_id: Optional[str] = None*, *targets: Optional[List[str]] = None*, *queue_delay: float = 1.5*) → None

Send message for TTS at speaker.

This is the old method which used Alexa Simon Says which did not work for WHA. This will not beep prior to sending. send_announcement should be used instead.

Args: message (string): The message to speak. For canned messages, the message

must start with *alexa.cannedtts.speak* as discovered in the routines.

**customer_id (string): CustomerId to use for authorization. When none** specified this defaults to the logged in user. Used with households where others may have their own music.

**targets (list(string)): WARNING: This is currently non functional due** to Alexa's API and is only included for future proofing. List of serialNumber or accountName to send the tts to. Only those in this AlexaAPI account will be searched. If None, announce will be self.

**queue_delay (float, optional): The number of seconds to wait** for commands to queue together. Defaults to 1.5. Must be positive.

**async set_background**(*url: str*) → bool

Set background for Echo Show.

Sets the background to Alexa App Photo with the specific https url.

Args url (URL): valid https url for the image

Returns Whether the command was successful.

**async set_bluetooth**(*mac: str*) → None

Pair with bluetooth device with mac address.

**async set_dnd_state**(*state: bool*) → None

Set Do Not Disturb state.

Args: state (boolean): true or false

Returns json

**async set_guard_state**(*entity_id: str*, *state: str*, *queue_delay: float = 1.5*) → None

Set Guard state.

Args: entity_id (Text): numeric ending of applianceId of RedRock Panel state (Text): AWAY, HOME queue_delay (float, optional): The number of seconds to wait

for commands to queue together. Defaults to 1.5. Must be positive.

Returns json

**async static set_light_state**(*login:* alexapy.alexalogin.AlexaLogin, *entity_id: str*, *power_on: bool = True*, *brightness: Optional[int] = None*, *color_name: Optional[str] = None*, *color_temperature_name: Optional[str] = None*) → Optional[Dict[str, Any]]

Set state of a light.

Args: login (AlexaLogin): Successfully logged in AlexaLogin entity_id (Text): Entity ID of The light. Not the Application ID. power_on (bool): Should the light be on or off. brightness (Optional[int]): 0-100 or None to leave as is color_name (Optional[Text]): The name of a color that Alexa supports in snake case. color_temperature_name (Optional[Text]): The name of a color temperature name that Alexa supports in snake case.

Returns json

**async set_media**(*data: Dict[str, Any]*) → None

Select the media player.

**async static set_notifications**(*login:* alexapy.alexalogin.AlexaLogin, *data*) → Optional[Dict[str, Any]]

Update Alexa notification.

Args: login (AlexaLogin): Successfully logged in AlexaLogin data : Data to pass to notifications

Returns json

**async set_volume**(*volume: float*, *customer_id: Optional[str] = None*, *queue_delay: float = 1.5*) → None

Set volume.

Args: volume (float): The volume between 0 and 1. customer_id (string): CustomerId to use for sending. When none

specified this defaults to the logged in user.

**queue_delay (float, optional): The number of seconds to wait** for commands to queue together. Defaults to 1.5. Must be positive.

**async shuffle**(*setting: bool*) → None

Shuffle.

setting (string) : true or false

**async static static_set_guard_state**(*login:* alexapy.alexalogin.AlexaLogin, *entity_id: str*, *state: str*) → Optional[Dict[str, Any]]

Set state of Alexa guard.

Args: login (AlexaLogin): Successfully logged in AlexaLogin entity_id (Text): entityId of RedRock Panel state (Text): ARMED_AWAY, ARMED_STAY

Returns json

**async stop**(*customer_id: Optional[str] = None*, *queue_delay: float = 1.5*, *all_devices: bool = False*) → None

Stop device playback.

**Keyword Arguments**

- **(default** (*all_devices {bool} -- Whether all devices should be stopped*) – {None})
- **wait** (*queue_delay {float} -- The number of seconds to*) – for commands to queue together. Must be positive. (default: {1.5})
- **(default** – {False})

**update_login**(*login:* alexapy.alexalogin.AlexaLogin) → bool

Update Login if it has changed.

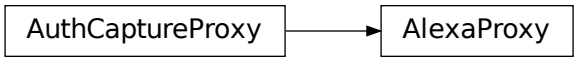
**Args** login (AlexaLogin): AlexaLogin to check

**Returns** bool: True if change detected

**class** alexapy.**AlexaProxy**(*login:* alexapy.alexalogin.AlexaLogin, *base_url: str*)

Class to handle proxy login connections to Alexa.

### Inheritance



**autofill**(*items: dict*, *html: str*) → str

Autofill input tags in form in html.

**Args** html (Text): html to convert items (dict): Dictionary of values to fill

**Returns** Text: html with values filled in

**change_login**(*login:* alexapy.alexalogin.AlexaLogin) → None

Change login.

**Parameters** **login** (AlexaLogin) – AlexaLogin object to update after completion of proxy.

**async test_amazon_url**(*resp*, *data*, *query*) → Optional[Union[yarl.URL, str]]

Test for Alexa success.

**Args** resp (httpx.Response): The aiohttp response. data (Dict[Text, Any]): Dictionary of all post data captured through proxy with overwrites for duplicate keys. query (Dict[Text, Any]): Dictionary of all query data with overwrites for duplicate keys.

**Returns** Optional[Union[URL, Text]]: URL for a http 302 redirect or Text to display on success. None indicates test did not pass.

**class** alexapy.**WebsocketEchoClient**(*login:* alexapy.alexalogin.AlexaLogin, *msg_callback: Callable[[alexapy.alexawebsocket.Message], Coroutine[Any, Any, None]]*, *open_callback: Callable[[], Coroutine[Any, Any, None]]*, *close_callback: Callable[[], Coroutine[Any, Any, None]]*, *error_callback: Callable[[str], Coroutine[Any, Any, None]]*)

WebSocket Client Class for Echo Devices.

Based on code from openHAB: https://github.com/openhab/openhab2-addons/blob/master/addons/binding/org.openhab.binding.amazonechocontrol/src/main/java/org/openhab/binding/amazonechocontrol/internal/WebSocketConnection.java which is further based on: https://github.com/Apollon77/alexa-remote/blob/master/alexa-wsmqtt.js

### Inheritance


**async async_on_open**() → None

Handle Async WebSocket Open.

**async async_run**() → None

Start Async WebSocket Listener.

**on_close**(*future=''*) → None

Handle WebSocket Close.

**on_error**(*error: str = 'Unspecified'*) → None

Handle WebSocket Error.

**async on_message**(*message: bytes*) → None

Handle New Message.

**async process_messages**() → None

Start Async WebSocket Listener.

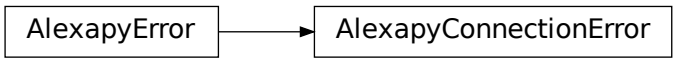
### 6.1.4 Exceptions

- *AlexapyConnectionError*: Define an error related to invalid requests.
- *AlexapyLoginCloseRequested*: Define an error related to requesting access to API after requested close.
- *AlexapyLoginError*: Define an error related to no longer being logged in.
- *AlexapyPyotpInvalidKey*: Define an error related to invalid 2FA key.

**exception** alexapy.**AlexapyConnectionError**

Define an error related to invalid requests.

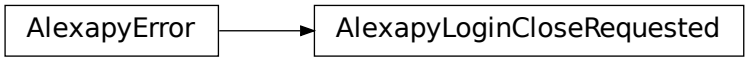
#### Inheritance




**exception** alexapy.**AlexapyLoginCloseRequested**

Define an error related to requesting access to API after requested close.

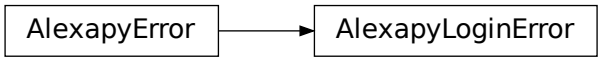
#### Inheritance




**exception** alexapy.**AlexapyLoginError**

Define an error related to no longer being logged in.

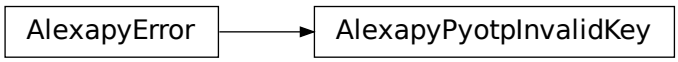
#### Inheritance




**exception** alexapy.**AlexapyPyotpInvalidKey**

Define an error related to invalid 2FA key.

#### Inheritance



CHAPTER

SEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## a

# INDEX