platypush Documentation

BlackLight

Mar 12, 2021

Contents:

Welcome to the Platypush reference of available plugins, backends and event types.

For more information on Platypush please check out:

- The Gitlab page of the project
- The online wiki for quickstart and examples
- The Blog articles for inspiration on use-cases possible projects

CHAPTER 1

Backends

1.1 platypush.backend.adafruit.io

class platypush.backend.adafruit.io.**AdafruitIoBackend**(*feeds*, **args*, ***kwargs*)

Backend that listens to messages received over the Adafruit IO message queue

Triggers:

- ***platypush.message.event.adafruit.ConnectedEvent*** **when the** backend connects to the Adafruit queue
- ***platypush.message.event.adafruit.DisconnectedEvent*** **when the** backend disconnects from the Adafruit queue
- ***platypush.message.event.adafruit.FeedUpdateEvent*** **when an** update event is received on a monitored feed

Requires:

- **The** ***platypush.plugins.adafruit.io.AdafruitIoPlugin*** **plugin to** be active and configured.

__init__(*feeds*, **args*, ***kwargs*)

Parameters **feeds** (*list[str]*) – List of feed IDs to monitor

on_message(*msg*)

Callback when a message is received on the backend. It parses and posts the message on the main bus. It should be called by the derived classes whenever a new message should be processed.

Parameters **msg** – Received message. It can be either a key-value dictionary, a platypush.message.Message object, or a string/byte UTF-8 encoded string

run()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.2 platypush.backend.alarm

class platypush.backend.alarm.**AlarmBackend**(*alarms: Union[list, Dict[str, Any], None] = None, audio_plugin: str = 'media.mplayer', *args, **kwargs*)

Backend to handle user-configured alarms.

Triggers:

- *platypush.message.event.alarm.AlarmStartedEvent* when an alarm starts.
- *platypush.message.event.alarm.AlarmSnoozedEvent* when an alarm is snoozed.
- *platypush.message.event.alarm.AlarmTimeoutEvent* when an alarm times out.
- *platypush.message.event.alarm.AlarmDismissedEvent* when an alarm is dismissed.

__init__(*alarms: Union[list, Dict[str, Any], None] = None, audio_plugin: str = 'media.mplayer', *args, **kwargs*)

Parameters alarms – List or name->value dict with the configured alarms. Example:

```yaml
morning_alarm:
    when: '0 7 * * 1-5'    # Cron expression format: run every weekday at 7 AM
    audio_file: ~/path/your_ringtone.mp3
    audio_plugin: media.mplayer
    audio_volume: 10       # 10%
    snooze_interval: 300    # 5 minutes snooze
    actions:
        - action: tts.say
          args:
              text: Good morning

        - action: light.hue.bri
          args:
              value: 1

        - action: light.hue.bri
          args:
              value: 140
              transitiontime: 150

one_shot_alarm:
    when: '2020-02-18T07:00:00.000000'    # One-shot execution, with timestamp␣
↪in ISO format
    audio_file: ~/path/your_ringtone.mp3
    actions:
        - action: light.hue.on
```

Parameters audio_plugin – Media plugin (instance of *platypush.plugins.media.MediaPlugin*) that will be used to play the alarm audio (default: `media.mplayer`).

class platypush.backend.alarm.**AlarmState**

An enumeration.

1.3 `platypush.backend.assistant`

class platypush.backend.assistant.**AssistantBackend**(*tts_plugin: Optional[str] = None, tts_args: Optional[Dict[str, Any]] = None, **kwargs*)

__init__(*tts_plugin: Optional[str] = None, tts_args: Optional[Dict[str, Any]] = None, **kwargs*)
Default assistant backend constructor.

Parameters

- **tts_plugin** – If set, and if the assistant returns the processed response as text, then the processed response will be played through the selected text-to-speech plugin (can be e.g. "tts", "tts.google" or any other implementation of *platypush.plugins.tts.TtsPlugin*).
- **tts_args** – Extra parameters to pass to the say method of the selected TTS plugin (e.g. language, voice or gender).

1.4 `platypush.backend.assistant.google`

class platypush.backend.assistant.google.**AssistantGoogleBackend**(*credentials_file='/home/docs/.config/go oauthlib-tool/credentials.json', device_model_id='Platypush', **kwargs*)

Google Assistant backend.

It listens for voice commands and post conversation events on the bus.

WARNING: The Google Assistant library used by this backend has officially been deprecated: https://developers.google.com/assistant/sdk/reference/library/python/. This backend still works on most of the devices where I use it, but its correct functioning is not guaranteed as the assistant library is no longer maintained.

Triggers:

- *platypush.message.event.assistant.ConversationStartEvent* when a new conversation starts
- *platypush.message.event.assistant.SpeechRecognizedEvent* when a new voice command is recognized
- platypush.message.event.assistant.NoResponse when a conversation returned no response
- *platypush.message.event.assistant.ResponseEvent* when the assistant is speaking a response
- *platypush.message.event.assistant.ConversationTimeoutEvent* when a conversation times out
- *platypush.message.event.assistant.ConversationEndEvent* when a new conversation ends
- *platypush.message.event.assistant.AlarmStartedEvent* when an alarm starts
- *platypush.message.event.assistant.AlarmEndEvent* when an alarm ends
- *platypush.message.event.assistant.TimerStartedEvent* when a timer starts

- *platypush.message.event.assistant.TimerEndEvent* when a timer ends
- *platypush.message.event.assistant.MicMutedEvent* when the microphone is muted.
- *platypush.message.event.assistant.MicUnmutedEvent* when the microphone is unmuted.

Requires:

- **google-assistant-library** (`pip install google-assistant-library`)
- **google-assistant-sdk[samples]** (`pip install google-assistant-sdk[samples]`)

`__init__`(*credentials_file='/home/docs/.config/google-oauthlib-tool/credentials.json', device_model_id='Platypush', **kwargs*)

Parameters

- **`credentials_file`** (*str*) – Path to the Google OAuth credentials file (default: ~/.config/google-oauthlib-tool/credentials.json). See https://developers.google.com/assistant/sdk/guides/library/python/embed/install-sample#generate_credentials for instructions to get your own credentials file.
- **`device_model_id`** (*str*) – Device model ID to use for the assistant (default: Platypush)

`run`()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

`start_conversation`()
Starts an assistant conversation

`stop_conversation`()
Stops an assistant conversation

1.5 platypush.backend.assistant.snowboy

class platypush.backend.assistant.snowboy.**AssistantSnowboyBackend**(*models, audio_gain=1.0, **kwargs*)

Backend for detecting custom voice hotwords through Snowboy. The purpose of this component is only to detect the hotword specified in your Snowboy voice model. If you want to trigger proper assistant conversations or custom speech recognition, you should create a hook in your configuration on HotwordDetectedEvent to trigger the conversation on whichever assistant plugin you're using (Google, Alexa…)

Triggers:

- *platypush.message.event.assistant.HotwordDetectedEvent* whenever the hotword has been detected

Requires:

- **snowboy** (`pip install snowboy`)

Manual installation for snowboy and its Python bindings if the command above fails:

```
$ [sudo] apt-get install libatlas-base-dev swig
$ [sudo] pip install pyaudio
$ git clone https://github.com/Kitt-AI/snowboy
$ cd snowboy/swig/Python3
$ make
```

(continues on next page)

(continued from previous page)

```
$ cd ../..
$ python3 setup.py build
$ [sudo] python setup.py install
```

You will also need a voice model for the hotword detection. You can find some under the `resources/models` directory of the Snowboy repository, or train/download other models from https://snowboy.kitt.ai.

`__init__`(*models*, *audio_gain=1.0*, ***kwargs*)

Parameters

- **`models`** (`dict`) – Map (name -> configuration) of voice models to be used by the assistant. See https://snowboy.kitt.ai/ for training/downloading models. Sample format:

```
ok_google:    # Hotword model name
    voice_model_file: /path/models/OK Google.pmdl  # Voice model␣
↪file location
    sensitivity: 0.5            # Model sensitivity, between 0␣
↪and 1 (default: 0.5)
    assistant_plugin: assistant.google.pushtotalk  # When the␣
↪hotword is detected trigger the Google
                                                   # push-to-talk␣
↪assistant plugin (optional)
    assistant_language: en-US   # The assistant will conversate␣
↪in English when this hotword is
        detected (optional)
    detect_sound: /path/to/bell.wav   # Sound file to be played␣
↪when the hotword is detected (optional)

ciao_google:  # Hotword model name
    voice_model_file: /path/models/Ciao Google.pmdl  # Voice␣
↪model file location
    sensitivity: 0.5            # Model sensitivity, between 0␣
↪and 1 (default: 0.5)
    assistant_plugin: assistant.google.pushtotalk    # When the␣
↪hotword is detected trigger the Google
                                                     # push-to-
↪talk assistant plugin (optional)
    assistant_language: it-IT   # The assistant will conversate␣
↪in Italian when this hotword is
                                # detected (optional)
    detect_sound: /path/to/bell.wav   # Sound file to be played␣
↪when the hotword is detected (optional)
```

- **`audio_gain`** (`float`) – Audio gain, between 0 and 1. Default: 1

`hotword_detected`(*hotword*)
Callback called on hotword detection

`on_stop`()
Callback invoked when the process stops

`run`()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.6 platypush.backend.bluetooth

class `platypush.backend.bluetooth.`**`BluetoothBackend`**(*address: str = ”, port: int = None, directory: str = None, whitelisted_addresses=None, **kwargs*)

`__init__`(*address: str = ”, port: int = None, directory: str = None, whitelisted_addresses=None, **kwargs*)

Parameters

- **`bus`** – Reference to the bus object to be used in the backend
- **`poll_seconds`** – If the backend implements a `loop` method, this parameter expresses how often the loop should run in seconds.
- **`kwargs`** – Key-value configuration for the backend

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn’t defined.

`stop`()

Stops the backend thread by sending a STOP event on its bus

1.7 platypush.backend.bluetooth.fileserver

class `platypush.backend.bluetooth.fileserver.`**`BluetoothFileserverBackend`**(*port: int, address: str = ”, directory: str = ’/home/docs’, whitelisted_addresses: list = None, **kwargs*)

Bluetooth OBEX file server. Enable it to allow bluetooth devices to browse files on this machine.

If you run platypush as a non-root user (and you should) then you to change the group owner of the service discovery protocol file (/var/run/sdp) and add your user to that group. See here for details.

Requires:

- **pybluez** (`pip install pybluez`)
- **pyobex** (`pip install git+https://github.com/BlackLight/PyOBEX`)

`__init__`(*port: int, address: str = ”, directory: str = ’/home/docs’, whitelisted_addresses: list = None, **kwargs*)

Parameters

- **`port`** – Bluetooth listen port
- **`address`** – Bluetooth address to bind the server to (default: any)
- **`directory`** – Directory to share (default: HOME directory)
- **`whitelisted_addresses`** – If set then only accept connections from the listed device addresses

1.8 `platypush.backend.bluetooth.pushserver`

class `platypush.backend.bluetooth.pushserver.`**`BluetoothPushserverBackend`**(*port: int*, *address: str = ''*, *directory: str = '/home/docs/bluetooth'*, *whitelisted_addresses: list = None*, ***kwargs*)

Bluetooth OBEX push server. Enable it to allow bluetooth file transfers from other devices.

If you run platypush as a non-root user (and you should) then you to change the group owner of the service discovery protocol file (/var/run/sdp) and add your user to that group. See here for details.

Requires:

- **pybluez** (`pip install pybluez`)
- **pyobex** (`pip install git+https://github.com/BlackLight/PyOBEX`)

`__init__`(*port: int*, *address: str = ''*, *directory: str = '/home/docs/bluetooth'*, *whitelisted_addresses: list = None*, ***kwargs*)

Parameters

- **`port`** – Bluetooth listen port
- **`address`** – Bluetooth address to bind the server to (default: any)
- **`directory`** – Destination directory where files will be downloaded (default: ~/bluetooth)
- **`whitelisted_addresses`** – If set then only accept connections from the listed device addresses

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.9 `platypush.backend.bluetooth.scanner`

class `platypush.backend.bluetooth.scanner.`**`BluetoothScannerBackend`**(*device_id: Optional[int] = None*, *scan_duration: int = 10*, *track_devices: Optional[List[str]] = None*, ***kwargs*)

This backend periodically scans for available bluetooth devices and returns events when a devices enter or exits the range.

Triggers:

- *`platypush.message.event.bluetooth.BluetoothDeviceFoundEvent`* when a new bluetooth device is found.
- *`platypush.message.event.bluetooth.BluetoothDeviceLostEvent`* when a bluetooth device is lost.

Requires:

- The *`platypush.plugins.bluetooth.BluetoothPlugin`* plugin working.

`__init__`(*device_id: Optional[int] = None*, *scan_duration: int = 10*, *track_devices: Optional[List[str]] = None*, ***kwargs*)

Parameters

- **`device_id`** – Bluetooth adapter ID to use (default configured on the `bluetooth` plugin if None).
- **`scan_duration`** – How long the scan should run (default: 10 seconds).
- **`track_devices`** – List of addresses of devices to actively track, even if they aren't discoverable.

`get_measurement`()

Wrapper around `plugin.get_measurement()` that can filter events on specified enabled sensors data or on specified tolerance values. It can be overridden by derived classes.

`on_stop`()

Callback invoked when the process stops

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.10 `platypush.backend.bluetooth.scanner.ble`

class `platypush.backend.bluetooth.scanner.ble.`**`BluetoothBleScannerBackend`**(*interface: Optional[int] = None*, *scan_duration: int = 10*, ***kwargs*)

This backend periodically scans for available bluetooth low-energy devices and returns events when a devices enter or exits the range.

Triggers:

- *`platypush.message.event.bluetooth.BluetoothDeviceFoundEvent`* when a new bluetooth device is found.
- *`platypush.message.event.bluetooth.BluetoothDeviceLostEvent`* when a bluetooth device is lost.

Requires:

- The `platypush.plugins.bluetooth.BluetoothBlePlugin` plugin working.

`__init__`(*interface: Optional[int] = None*, *scan_duration: int = 10*, ***kwargs*)

Parameters

- **`interface`** – Bluetooth adapter name to use (default configured on the `bluetooth.ble` plugin if None).
- **`scan_duration`** – How long the scan should run (default: 10 seconds).

1.11 `platypush.backend.button.flic`

class `platypush.backend.button.flic.`**`ButtonFlicBackend`**(*server='localhost'*, *long_press_timeout=0.3*, *btn_timeout=0.5*, ***kwargs*)

Backend that listen for events from the Flic (https://flic.io/) bluetooth smart buttons.

Triggers:

- *`platypush.message.event.button.flic.FlicButtonEvent`* when a button is pressed. The event will also contain the press sequence (e.g. `["ShortPressEvent", "LongPressEvent", "ShortPressEvent"]`)

Requires:

- **fliclib** (https://github.com/50ButtonsEach/fliclib-linux-hci). For the backend to work properly you need to have the `flicd` daemon from the fliclib running, and you have to first pair the buttons with your device using any of the scanners provided by the library.

`__init__`(*server='localhost'*, *long_press_timeout=0.3*, *btn_timeout=0.5*, ***kwargs*)

Parameters

- **`server`** (*`str`*) – flicd server host (default: localhost)

- **long_press_timeout** (*float*) – How long you should press a button for a press action to be considered "long press" (default: 0.3 secohds)
- **btn_timeout** (*float*) – How long since the last button release before considering the user interaction completed (default: 0.5 seconds)

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.12 platypush.backend.camera.pi

class platypush.backend.camera.pi.**CameraPiBackend**(*listen_port*, *x_resolution=640*, *y_resolution=480*, *redis_queue='platypush/camera/pi'*, *start_recording_on_startup=True*, *framerate=24*, *hflip=False*, *vflip=False*, *sharpness=0*, *contrast=0*, *brightness=50*, *video_stabilization=False*, *iso=0*, *exposure_compensation=0*, *exposure_mode='auto'*, *meter_mode='average'*, *awb_mode='auto'*, *image_effect='none'*, *color_effects=None*, *rotation=0*, *crop=(0.0, 0.0, 1.0, 1.0)*, ***kwargs*)

Backend to interact with a Raspberry Pi camera. It can start and stop recordings and take pictures. It can be programmatically controlled through the `platypush.plugins.camera.pi` plugin. Note that the Redis backend must be configured and running to enable camera control.

Requires:

- **picamera** (`pip install picamera`)
- **redis** (`pip install redis`) for inter-process communication with the camera process

This backend is **DEPRECATED**. Use the plugin `platypush.plugins.camera.pi.CameraPiPlugin` instead to run Pi camera actions. If you want to start streaming the camera on application start then simply create an event hook on `platypush.message.event.application.ApplicationStartedEvent` that runs `camera.pi.start_streaming`.

class CameraAction
An enumeration.

__init__(*listen_port*, *x_resolution=640*, *y_resolution=480*, *redis_queue='platypush/camera/pi'*, *start_recording_on_startup=True*, *framerate=24*, *hflip=False*, *vflip=False*, *sharpness=0*, *contrast=0*, *brightness=50*, *video_stabilization=False*, *iso=0*, *exposure_compensation=0*, *exposure_mode='auto'*, *meter_mode='average'*, *awb_mode='auto'*, *image_effect='none'*, *color_effects=None*, *rotation=0*, *crop=(0.0, 0.0, 1.0, 1.0)*, ***kwargs*)
See https://www.raspberrypi.org/documentation/usage/camera/python/README.md for a detailed reference about the Pi camera options.

Parameters **listen_port** (*int*) – Port where the camera process will provide the video output while recording

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

start_recording(*video_file=None, format='h264'*)
Start a recording.

Parameters

- **video_file** (*str*) – Output video file. If specified, the video will be recorded to file, otherwise it will be served via TCP/IP on the listen_port. Use `stop_recording` to stop the recording.
- **format** (*str*) – Video format (default: h264)

stop_recording()
Stops recording

take_picture(*image_file*)
Take a picture.

Parameters **image_file** (*str*) – Output image file

1.13 `platypush.backend.chat.telegram`

class platypush.backend.chat.telegram.**ChatTelegramBackend**(*authorized_chat_ids: Optional[List[Union[str, int]]] = None, **kwargs*)

Telegram bot that listens for messages and updates.

Triggers:

- *platypush.message.event.chat.telegram.TextMessageEvent* when a text message is received.
- *platypush.message.event.chat.telegram.PhotoMessageEvent* when a photo is received.
- *platypush.message.event.chat.telegram.VideoMessageEvent* when a video is received.
- *platypush.message.event.chat.telegram.LocationMessageEvent* when a location is received.
- *platypush.message.event.chat.telegram.ContactMessageEvent* when a contact is received.
- *platypush.message.event.chat.telegram.DocumentMessageEvent* when a document is received.
- *platypush.message.event.chat.telegram.CommandMessageEvent* when a command message is received.
- `platypush.message.event.chat.telegram.GroupCreatedEvent` when the bot is invited to a new group.

Requires:

- The *platypush.plugins.chat.telegram.ChatTelegramPlugin* plugin configured

__init__(*authorized_chat_ids: Optional[List[Union[str, int]]] = None, **kwargs*)

Parameters **authorized_chat_ids** – Optional list of chat_id/user_id which are authorized to send messages to the bot. If nothing is specified then no restrictions are applied.

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.14 platypush.backend.clipboard

class platypush.backend.clipboard.**ClipboardBackend**(**args*, ***kwargs*)
This backend monitors for changes in the clipboard and generates even when the user copies a new text.

Requires:

- **pyperclip** (`pip install pyperclip`)

Triggers:

- *platypush.message.event.clipboard.ClipboardEvent* on clipboard update.

__init__(**args*, ***kwargs*)

Parameters

- **bus** – Reference to the bus object to be used in the backend
- **poll_seconds** – If the backend implements a `loop` method, this parameter expresses how often the loop should run in seconds.
- **kwargs** – Key-value configuration for the backend

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.15 platypush.backend.covid19

class platypush.backend.covid19.**Covid19Backend**(*country: Union[str, List[str], None], poll_seconds: Optional[float] = 3600.0, **kwargs*)
This backend polls new data about the Covid-19 pandemic diffusion and triggers events when new data is available.

Triggers:

- *platypush.message.event.covid19.Covid19UpdateEvent* when new data is available.

__init__(*country: Union[str, List[str], None], poll_seconds: Optional[float] = 3600.0, **kwargs*)

Parameters country – Default country (or list of countries) to retrieve the stats for. It can either be the full country name or the country code. Special values:

- `world`: Get worldwide stats.
- `all`: Get all the available stats.

Default: either the default configured on the *platypush.plugins.covid19.Covid19Plugin* plugin or `world`.

Parameters poll_seconds – How often the backend should check for new check-ins (default: one hour).

class platypush.backend.covid19.**Covid19Update**(***kwargs*)
Models the Covid19Data table

`__init__`(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

1.16 `platypush.backend.dbus`

class `platypush.backend.dbus.`**`DBusService`**(**args*, ***kwargs*)

`Post`(*msg: dict*)

This method accepts a message as a dictionary (either representing a valid request or an event) and either executes it (request) or forwards it to the application bus (event).

Parameters **`msg`** – Request or event, as a dictionary.

Returns The return value of the request, or 0 if the message is an event.

class `platypush.backend.dbus.`**`DbusBackend`**(*bus_name='org.platypush.Bus'*, *service_path='/MessageService'*, **args*, ***kwargs*)

This backend acts as a proxy that receives messages (requests or events) on the DBus and forwards them to the application bus.

The name of the messaging interface exposed by Platypush is `org.platypush.MessageBusInterface` and it exposes `Post` method, which accepts a dictionary representing a valid Platypush message (either a request or an event) and either executes it or forwards it to the application bus.

Requires:

- **dbus-python** (`pip install dbus-python`)

`__init__`(*bus_name='org.platypush.Bus'*, *service_path='/MessageService'*, **args*, ***kwargs*)

Parameters

- **`bus_name`** – Name of the bus where the application will listen for incoming messages (default: `org.platypush.Bus`).
- **`service_path`** – Path to the service exposed by the app (default: `/MessageService`).

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.17 `platypush.backend.foursquare`

class `platypush.backend.foursquare.`**`FoursquareBackend`**(*poll_seconds: Optional[float] = 60.0*, **args*, ***kwargs*)

This backend polls for new check-ins on the user's Foursquare account and triggers an event when a new check-in occurs.

Requires:

- The *`platypush.plugins.foursquare.FoursquarePlugin`* plugin configured and enabled.

Triggers:

- *platypush.message.event.foursquare.FoursquareCheckinEvent* when a new check-in occurs.

__init__(*poll_seconds: Optional[float] = 60.0*, **args*, ***kwargs*)

Parameters **poll_seconds** – How often the backend should check for new check-ins (default: one minute).

1.18 platypush.backend.github

class platypush.backend.github.**GithubBackend**(*user: str*, *user_token: str*, *repos: Optional[List[str]] = None*, *org: Optional[str] = None*, *poll_seconds: int = 60*, *max_events_per_scan: Optional[int] = 10*, **args*, ***kwargs*)

This backend monitors for notifications and events either on Github user, organization or repository level. You'll need a Github personal access token to use the service. To get one:

- Access your Github profile settings
- Select *Developer Settings*
- Select *Personal access tokens*
- Click *Generate new token*

This backend requires the following permissions:

- `repo`
- `notifications`
- `read:org` if you want to access repositories on organization level.

Triggers:

- *platypush.message.event.github.GithubPushEvent* when a new push is created.
- *platypush.message.event.github.GithubCommitCommentEvent* when a new commit comment is created.
- *platypush.message.event.github.GithubCreateEvent* when a tag or branch is created.
- *platypush.message.event.github.GithubDeleteEvent* when a tag or branch is deleted.
- *platypush.message.event.github.GithubForkEvent* when a user forks a repository.
- *platypush.message.event.github.GithubWikiEvent* when new activity happens on a repository wiki.
- *platypush.message.event.github.GithubIssueCommentEvent* when new activity happens on an issue comment.
- *platypush.message.event.github.GithubIssueEvent* when new repository issue activity happens.
- *platypush.message.event.github.GithubMemberEvent* when new repository collaborators activity happens.
- *platypush.message.event.github.GithubPublicEvent* when a repository goes public.
- *platypush.message.event.github.GithubPullRequestEvent* when new pull request related activity happens.

- ***platypush.message.event.github.GithubPullRequestReviewCommentEvent* when activity happens**
  request commit.
- *platypush.message.event.github.GithubReleaseEvent* when a new release happens.
- *platypush.message.event.github.GithubSponsorshipEvent* when new sponsorship related activity happens.
- *platypush.message.event.github.GithubWatchEvent* when someone stars/starts watching a repository.
- ***platypush.message.event.github.GithubEvent* for any event that doesn't fall in the above categories**
  (`event_type` will be set accordingly).

`__init__`(*user: str*, *user_token: str*, *repos: Optional[List[str]] = None*, *org: Optional[str] = None*, *poll_seconds: int = 60*, *max_events_per_scan: Optional[int] = 10*, **args*, ***kwargs*)

If neither `repos` nor `org` is specified then the backend will monitor all new events on user level.

Parameters

- **`user`** – Github username.
- **`user_token`** – Github personal access token.
- **`repos`** – List of repos to be monitored - if a list is provided then only these repositories will be monitored for events. Repositories should be passed in the format `username/repository`.
- **`org`** – Organization to be monitored - if provided then only this organization will be monitored for events.
- **`poll_seconds`** – How often the backend should check for new events, in seconds (default: 60).
- **`max_events_per_scan`** – Maximum number of events per resource that will be triggered if there is a large number of events/notification since the last check (default: 10). Specify 0 or null for no limit.

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

class `platypush.backend.github.`**`GithubResource`**(***kwargs*)

Models the GithubLastEvent table, containing the timestamp where a certain URL was last checked.

`__init__`(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

1.19 `platypush.backend.google.fit`

class `platypush.backend.google.fit.`**`GoogleFitBackend`**(*data_sources*, *user_id='me'*, *poll_seconds=60*, **args*, ***kwargs*)

This backend will listen for new Google Fit events (e.g. new weight/height measurements, new fitness activities etc.) on the specified data streams and fire an event upon new data.

Triggers:

- **`platypush.message.event.google.fit.GoogleFitEvent`** when a new data point is received on one of the registered streams.

Requires:

- **The google.fit plugin** (`platypush.plugins.google.fit.GoogleFitPlugin`) enabled.
- The **db** plugin (`platypush.plugins.db`) configured

`__init__`(*data_sources*, *user_id='me'*, *poll_seconds=60*, **args*, ***kwargs*)

Parameters

- **`data_sources`** (*`list[str]`*) – Google Fit data source IDs to monitor. You can get a list of the available data sources through the `platypush.plugins.google.fit.get_data_sources()` action
- **`user_id`** (*`str`*) – Google user ID to track (default: 'me')
- **`poll_seconds`** (*`float`*) – How often the backend will query the data sources for new data points (default: 60 seconds)

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.20 `platypush.backend.google.pubsub`

class `platypush.backend.google.pubsub.`**`GooglePubsubBackend`**(*topics: List[str], credentials_file: Optional[str] = None, *args, **kwargs*)

Subscribe to a list of topics on a Google Pub/Sub instance. See `platypush.plugins.google.pubsub.GooglePubsubPlugin` for a reference on how to generate your project and credentials file.

Triggers:

- `platypush.message.event.google.pubsub.GooglePubsubMessageEvent` when a new message is received on a subscribed topic.

Requires:

- **google-cloud-pubsub** (`pip install google-cloud-pubsub`)

`__init__`(*topics: List[str], credentials_file: Optional[str] = None, *args, **kwargs*)

Parameters

- **`topics`** – List of topics to subscribe. You can either specify the full topic name in the format `projects/<project_id>/topics/<topic_name>`, where `<project_id>` must be the ID of your Google Pub/Sub project, or just `<topic_name>` - in such case it's implied that you refer to the `topic_name` under the `project_id` of your service credentials.
- **`credentials_file`** – Path to the Pub/Sub service credentials file (default: value configured on the `google.pubsub` plugin or `~/.credentials/platypush/google/pubsub.json`).

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.21 `platypush.backend.gps`

class `platypush.backend.gps.`**`GpsBackend`**(*gpsd_server='localhost'*, *gpsd_port=2947*, ***kwargs*)

This backend can interact with a GPS device and listen for events.

Triggers:

- *`platypush.message.event.gps.GPSVersionEvent`* when a GPS device advertises its version data
- *`platypush.message.event.gps.GPSDeviceEvent`* when a GPS device is connected or updated
- *`platypush.message.event.gps.GPSUpdateEvent`* when a GPS device has new data

Requires:

- **gps** (`pip install gps`)
- **gpsd** daemon running (`apt-get install gpsd` or `pacman -S gpsd` depending on your distro)

Once installed gpsd you need to run it and associate it to your device. Example if your GPS device communicates over USB and is available on /dev/ttyUSB0:

```
[sudo] gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
```

The best option is probably to run gpsd at startup as a systemd service.

`__init__`(*gpsd_server='localhost'*, *gpsd_port=2947*, ***kwargs*)

Parameters

- **`gpsd_server`** (*`str`*) – gpsd daemon server name/address (default: localhost)
- **`gpsd_port`** (*`int or str`*) – Port of the gpsd daemon (default: 2947)

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.22 `platypush.backend.http`

class `platypush.backend.http.`**`HttpBackend`**(*port=8008*, *websocket_port=8009*, *bind_address='0.0.0.0'*, *disable_websocket=False*, *resource_dirs=None*, *ssl_cert=None*, *ssl_key=None*, *ssl_cafile=None*, *ssl_capath=None*, *maps=None*, *run_externally=False*, *uwsgi_args=None*, ***kwargs*)

The HTTP backend is a general-purpose web server.

Example configuration:

```
backend.http:
    # Default HTTP listen port
    port: 8008
    # Default websocket port
    websocket_port: 8009
    # External folders that will be exposed over `/resources/<name>`
```

(continues on next page)

(continued from previous page)

```
    resource_dirs:
        photos: /mnt/hd/photos
        videos: /mnt/hd/videos
        music: /mnt/hd/music
```

You can leverage this backend:

- To execute Platypush commands via HTTP calls. In order to do so:
  - Register a user to Platypush through the web panel (usually served on `http://host:8008/`).
  - Generate a token for your user, either through the web panel (Settings -> Generate Token) or via API:

    ```
    curl -XPOST -H 'Content-Type: application/json' -d '
      {
        "username": "$YOUR_USER",
        "password": "$YOUR_PASSWORD"
      }' http://host:8008/auth
    ```

  - Execute actions through the `/execute` endpoint:

    ```
    curl -XPOST -H 'Content-Type: application/json' -H "Authorization:␣
    ↪Bearer $YOUR_TOKEN" -d '
      {
        "type": "request",
        "action": "tts.say",
        "args": {
          "text": "This is a test"
        }
      }' http://host:8008/execute
    ```

- To interact with your system (and control plugins and backends) through the Platypush web panel, by default available on `http://host:8008/`. Any configured plugin that has an available panel plugin will be automatically added to the web panel.
- To display a fullscreen dashboard with custom widgets.
  - Widgets are available as Vue.js components under `platypush/backend/http/webapp/src/components/widgets`.
  - Explore their options (some may require some plugins or backends to be configured in order to work) and create a new dashboard template under `~/.config/platypush/dashboards`- e.g. `main.xml`:

    ```
    <Dashboard>
        <!-- Display the following widgets on the same row. Each row␣
    ↪consists of 12 columns.
             You can specify the width of each widget either through␣
    ↪class name (e.g. col-6 means
             6 columns out of 12, e.g. half the size of the row) or␣
    ↪inline style
             (e.g. `style="width: 50%"`). -->
        <Row>
            <!-- Show a calendar widget with the upcoming events. It␣
    ↪requires the `calendar` plugin to
                 be enabled and configured. -->
            <Calendar class="col-6" />
    ```

(continues on next page)

(continued from previous page)

```
        <!-- Show the current track and other playback info. It␣
↪requires `music.mpd` plugin or any
             other music plugin enabled. -->
        <Music class="col-3" />

        <!-- Show current date, time and weather. It requires a␣
↪`weather` plugin or backend enabled -->
        <DateTimeWeather class="col-3" />
    </Row>

    <!-- Display the following widgets on a second row -->
    <Row>
        <!-- Show a carousel of images from a local folder. For␣
↪security reasons, the folder must be
             explicitly exposed as an HTTP resource through the␣
↪backend `resource_dirs` attribute. -->
        <ImageCarousel class="col-6" img-dir="/mnt/hd/photos/
↪carousel" />

        <!-- Show the news headlines parsed from a list of RSS feed␣
↪and stored locally through the
             `http.poll` backend -->
        <RssNews class="col-6" db="sqlite:////path/to/your/rss.db" /
↪>
    </Row>
</Dashboard>
```

  – The dashboard will be accessible under `http://host:8008/dashboard/<name>`, where `name=main` if for example you stored your template under `~/.config/platypush/dashboards/main.xml`.

- To expose custom endpoints that can be called as web hooks by other applications and run some custom logic. All you have to do in this case is to create a hook on a *platypush.message.event.http.hook.WebhookEvent* with the endpoint that you want to expose and store it under e.g. `~/.config/platypush/scripts/hooks.py`:

```python
from platypush.context import get_plugin
from platypush.event.hook import hook
from platypush.message.event.http.hook import WebhookEvent

hook_token = 'abcdefabcdef'

# Expose the hook under the /hook/lights_toggle endpoint
@hook(WebhookEvent, hook='lights_toggle')
def lights_toggle(event, **context):
    # Do any checks on the request
    assert event.headers.get('X-Token') == hook_token, 'Unauthorized'

    # Run some actions
    lights = get_plugin('light.hue')
    lights.toggle()
```

Any plugin can register custom routes under `platypush/backend/http/app/routes/plugins`. Any additional route is managed as a Flask blueprint template and the *.py* module can expose lists of routes to the main webapp through the `__routes__` object (a list of Flask blueprints).

Security: Access to the endpoints requires at least one user to be registered. Access to the endpoints is regulated in the following ways (with the exception of event hooks, whose logic is up to the user):

- **Simple authentication** - i.e. registered username and password.
- **JWT token** provided either over as `Authorization: Bearer` header or `GET ?token=<TOKEN>` parameter. A JWT token can be generated either through the web panel or over the `/auth` endpoint.
- **Global platform token**, usually configured on the root of the `config.yaml` as `token: <VALUE>`. It can provided either over on the `X-Token` header or as a `GET ?token=<TOKEN>` parameter.
- **Session token**, generated upon login, it can be used to authenticate requests through the `Cookie` header (cookie name: `session_token`).

Requires:

- **flask** (`pip install flask`)
- **redis** (`pip install redis`)
- **websockets** (`pip install websockets`)
- **python-dateutil** (`pip install python-dateutil`)
- **magic (`pip install python-magic`), optional, for MIME type** support if you want to enable media streaming
- **uwsgi (`pip install uwsgi` plus uwsgi server installed on your** system if required) - optional but recommended. By default the Platypush web server will run in a process spawned on the fly by the HTTP backend. However, being a Flask app, it will serve clients in a single thread and won't support many features of a full-blown web server.

Base command to run the web server over uwsgi:

```
uwsgi --http :8008 --module platypush.backend.http.uwsgi --master --processes 4 --
↪threads 4
```

Bear in mind that the main webapp is defined in `platypush.backend.http.app:application` and the WSGI startup script is stored under `platypush/backend/http/uwsgi.py`.

`__init__`(*port=8008*, *websocket_port=8009*, *bind_address='0.0.0.0'*, *disable_websocket=False*, *resource_dirs=None*, *ssl_cert=None*, *ssl_key=None*, *ssl_cafile=None*, *ssl_capath=None*, *maps=None*, *run_externally=False*, *uwsgi_args=None*, ***kwargs*)

Parameters

- **port** (*int*) – Listen port for the web server (default: 8008)
- **websocket_port** (*int*) – Listen port for the websocket server (default: 8009)
- **bind_address** (*str*) – Address/interface to bind to (default: 0.0.0.0, accept connection from any IP)
- **disable_websocket** (*bool*) – Disable the websocket interface (default: False)
- **ssl_cert** (*str*) – Set it to the path of your certificate file if you want to enable HTTPS (default: None)
- **ssl_key** (*str*) – Set it to the path of your key file if you want to enable HTTPS (default: None)
- **ssl_cafile** (*str*) – Set it to the path of your certificate authority file if you want to enable HTTPS (default: None)
- **ssl_capath** (*str*) – Set it to the path of your certificate authority directory if you want to enable HTTPS (default: None)

- **resource_dirs** (*dict[str, str]*) – Static resources directories that will be accessible through `/resources/<path>`. It is expressed as a map where the key is the relative path under `/resources` to expose and the value is the absolute path to expose.
- **run_externally** (*bool*) – If set, then the HTTP backend will not directly spawn the web server. Set this option if you plan to run the webapp in a separate web server (recommended), like uwsgi or uwsgi+nginx.
- **uwsgi_args** (*list[str]*) – If `run_externally` is set and you would like the HTTP backend to directly spawn and control the uWSGI application server instance, then pass the list of uWSGI arguments through this parameter. Some examples include:

```
# Start uWSGI instance listening on HTTP port 8008 with 4
# processes
['--plugin', 'python', '--http-socket', ':8008', '--master', '--
↪processes', '4']

# Start uWSGI instance listening on uWSGI socket on port 3031.
# You can then use another full-blown web server, like nginx
# or Apache, to communicate with the uWSGI instance
['--plugin', 'python', '--socket', '127.0.0.1:3031', '--master',
↪'--processes', '4']
```

notify_web_clients (*event*)
Notify all the connected web clients (over websocket) of a new event

on_stop ()
On backend stop

run ()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

send_message (*msg*, ***kwargs*)
Sends a platypush.message.Message to a node. To be implemented in the derived classes. By default, if the Redis backend is configured then it will try to deliver the message to other consumers through the configured Redis main queue.

Parameters

- **msg** – The message to send
- **queue_name** – Send the message on a specific queue (default: the queue_name configured on the Redis backend)

websocket ()
Websocket main server

1.23 platypush.backend.http.poll

class platypush.backend.http.poll.**HttpPollBackend** (*requests*, **args*, ***kwargs*)
This backend will poll multiple HTTP endpoints/services and return events the bus whenever something new happened. Supported types: `platypush.backend.http.request.JsonHttpRequest` (for polling updates on a JSON endpoint), `platypush.backend.http.request.rss.RssUpdates` (for polling updates on an RSS feed). Example configuration:

```
backend.http.poll:
    requests:
```

(continues on next page)

(continued from previous page)

```
        -
            # Poll for updates on a JSON endpoint
            method: GET
            type: platypush.backend.http.request.JsonHttpRequest
            args:
                url: https://host.com/api/v1/endpoint
                headers:
                    Token: TOKEN
                params:
                    updatedSince: 1m
                timeout: 5  # Times out after 5 seconds (default)
            poll_seconds: 60  # Check for updates on this endpoint every 60␣
↪seconds (default)
            path: ${response['items']}  # Path in the JSON to check for new items.
                                        # Python expressions are supported.
                                        # Note that 'response' identifies the␣
↪JSON root.
                                        # Default value: JSON root.
        -
            # Poll for updates on an RSS feed
            type: platypush.backend.http.request.rss.RssUpdates
            url: http://www.theguardian.com/rss/world
            title: The Guardian - World News
            poll_seconds: 120
            max_entries: 10
```

Triggers: an update event for the relevant HTTP source if it contains new items. For example:

- *platypush.message.event.http.rss.NewFeedEvent* if a feed contains new items
- *platypush.message.event.http.HttpEvent* if a JSON endpoint contains new items

`__init__`(*requests*, **args*, ***kwargs*)

Parameters **requests** (*dict*) – Configuration of the requests to make (see class description for examples)

run()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.24 `platypush.backend.inotify`

class platypush.backend.inotify.**InotifyBackend**(*watch_paths=None*, ***kwargs*)

(Linux only) This backend will listen for events on the filesystem (whether a file/directory on a watch list is opened, modified, created, deleted, closed or had its permissions changed) and will trigger a relevant event.

Triggers:

- *platypush.message.event.inotify.InotifyCreateEvent* if a resource is created
- *platypush.message.event.inotify.InotifyAccessEvent* if a resource is accessed
- *platypush.message.event.inotify.InotifyOpenEvent* if a resource is opened
- *platypush.message.event.inotify.InotifyModifyEvent* if a resource is modified
- *platypush.message.event.inotify.InotifyPermissionsChangeEvent* if the permissions of a resource are changed

- `platypush.message.event.inotify.InotifyCloseEvent` if a resource is closed
- `platypush.message.event.inotify.InotifyDeleteEvent` if a resource is removed

Requires:

- **inotify** (`pip install inotify`)

`__init__`(*watch_paths=None*, ***kwargs*)

Parameters **`watch_paths`** (`str`) – Filesystem resources to watch for events

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.25 `platypush.backend.joystick`

class `platypush.backend.joystick.`**`JoystickBackend`**(*device*, **args*, ***kwargs*)

This backend will listen for events from a joystick device and post a JoystickEvent whenever a new event is captured.

Triggers:

- `platypush.message.event.joystick.JoystickEvent` when a new joystick event is received

Requires:

- **inputs** (`pip install inputs`)

`__init__`(*device*, **args*, ***kwargs*)

Parameters **`device`** – Path to the joystick device (e.g. */dev/input/js0*)

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.26 `platypush.backend.kafka`

class `platypush.backend.kafka.`**`KafkaBackend`**(*server='localhost:9092'*, *topic='platypush'*, ***kwargs*)

Backend to interact with an Apache Kafka (https://kafka.apache.org/) streaming platform, send and receive messages.

Requires:

- **kafka** (`pip install kafka-python`)

`__init__`(*server='localhost:9092'*, *topic='platypush'*, ***kwargs*)

Parameters

- **`server`** (`str`) – Kafka server name or address + port (default: `localhost:9092`)
- **`topic`** (`str`) – (Prefix) topic to listen to (default: platypush). The Platypush device_id (by default the hostname) will be appended to the topic (the real topic name will e.g. be "platypush.my_rpi")

`on_stop`()

Callback invoked when the process stops

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

send_message(*msg*)
Sends a platypush.message.Message to a node. To be implemented in the derived classes. By default, if the Redis backend is configured then it will try to deliver the message to other consumers through the configured Redis main queue.

Parameters

- **msg** – The message to send
- **queue_name** – Send the message on a specific queue (default: the queue_name configured on the Redis backend)

1.27 `platypush.backend.light.hue`

class `platypush.backend.light.hue.`**LightHueBackend**(*poll_seconds=10*, **args*, ***kwargs*)
This backend will periodically check for the status of your configured Philips Hue light devices and trigger events when the status of a device (power, saturation, brightness or hue) changes.

Triggers:

- *`platypush.message.event.light.LightStatusChangeEvent`* **when the** status of a lightbulb changes

Requires:

- **The** *`platypush.plugins.light.hue.LightHuePlugin`* **plugin to be** active and configured.

__init__(*poll_seconds=10*, **args*, ***kwargs*)

Parameters **poll_seconds** (*float*) – How often the backend will poll the Hue plugin for status updates. Default: 10 seconds

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.28 `platypush.backend.linode`

class `platypush.backend.linode.`**LinodeBackend**(*instances: Optional[List[str]] = None*, *poll_seconds: float = 30.0*, ***kwargs*)
This backend monitors the state of one or more Linode instances.

Triggers:

- *`platypush.message.event.linode.LinodeInstanceStatusChanged`* when the status of an instance changes.

Requires:

- The *`platypush.plugins.linode.LinodePlugin`* plugin configured.

__init__(*instances: Optional[List[str]] = None*, *poll_seconds: float = 30.0*, ***kwargs*)

Parameters **instances** – List of instances to monitor, by label (default: monitor all the instances).

1.29 `platypush.backend.local`

class `platypush.backend.local.`**`LocalBackend`**(*request_fifo*, *response_fifo*, ***kwargs*)

Sends and receive messages on two distinct local FIFOs, one for the requests and one for the responses. This is a legacy backend that should only be used for testing purposes.

You can use this backend either to send local commands to push through Pusher (or any other script), or debug. You can even send command on the command line and read the responses in this way:

```
# Send the request. Remember: the JSON must be all on one line.
$ cat <<EOF > /tmp/platypush-requests.fifo
{"type": "request", "target": "node_name", "action": "shell.exec", "args": {"cmd":
↪"echo ping"}, "origin": "node_name", "id": "put_an_id_here"}
EOF
$ cat /tmp/platypush-responses.fifo
ping
$
```

`__init__`(*request_fifo*, *response_fifo*, ***kwargs*)

Parameters

- **`bus`** – Reference to the bus object to be used in the backend
- **`poll_seconds`** – If the backend implements a `loop` method, this parameter expresses how often the loop should run in seconds.
- **`kwargs`** – Key-value configuration for the backend

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

`send_message`(*msg*, ***kwargs*)

Sends a platypush.message.Message to a node. To be implemented in the derived classes. By default, if the Redis backend is configured then it will try to deliver the message to other consumers through the configured Redis main queue.

Parameters

- **`msg`** – The message to send
- **`queue_name`** – Send the message on a specific queue (default: the queue_name configured on the Redis backend)

1.30 `platypush.backend.mail`

class `platypush.backend.mail.`**`MailBackend`**(*mailboxes: List[Dict[str, Any]], timeout: Optional[int] = 60, poll_seconds: Optional[int] = 60, **kwargs*)

This backend can subscribe to one or multiple mail servers and trigger events when new messages are received or messages are marked as seen.

It requires at least one plugin that extends *`platypush.plugins.mail.MailInPlugin`* (e.g. `mail.imap`) to be installed.

Triggers:

- *`platypush.message.event.mail.MailReceivedEvent`* when a new message is received.
- *`platypush.message.event.mail.MailSeenEvent`* when a message is marked as seen.

- *platypush.message.event.mail.MailFlaggedEvent* when a message is marked as flagged/starred.
- *platypush.message.event.mail.MailUnflaggedEvent* when a message is marked as unflagged/unstarred.

__init__(*mailboxes: List[Dict[str, Any]], timeout: Optional[int] = 60, poll_seconds: Optional[int] = 60, **kwargs*)

Parameters

- **mailboxes** – List of mailboxes to be monitored. Each mailbox entry contains a `plugin` attribute to identify the *platypush.plugins.mail.MailInPlugin* plugin that will be used (e.g. `mail.imap`) and the arguments that will be passed to `platypush.plugins.mail.MailInPlugin.search_unseen_messages()`. The `name` parameter can be used to identify this mailbox in the relevant events, otherwise `Mailbox #{id}` will be used as a name. Example configuration:

```
backend.mail:
    mailboxes:
        - plugin: mail.imap
          name: "My Local Server"
          username: me@mydomain.com
          password: my-imap-password
          server: localhost
          ssl: true
          folder: "All Mail"

        - plugin: mail.imap
          name: "GMail"
          username: me@gmail.com
          password: my-google-password
          server: imap.gmail.com
          ssl: true
           folder: "INBOX"
```

  If you have a default configuration available for a mail plugin you can implicitly reuse it without replicating it here. Example:

```
mail.imap:
    username: me@mydomain.com
    password: my-imap-password
    server: localhost
    ssl: true

backend.mail:
    mailboxes:
        # The mail.imap default configuration will be used
        - plugin: mail.imap
          name: "My Local Server"
          folder: "All Mail"
```

- **poll_seconds** – How often the backend should check the mail (default: 60).
- **timeout** – Connect/read timeout for a mailbox, in seconds (default: 60).

class platypush.backend.mail.**Mailbox**(*plugin: platypush.plugins.mail.MailInPlugin, name: str, args: dict*)

__init__(*plugin: platypush.plugins.mail.MailInPlugin, name: str, args: dict*) → None

class platypush.backend.mail.**MailboxStatus**(***kwargs*)

Models the MailboxStatus table, containing information about the state of a monitored mailbox.

__init__(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

1.31 platypush.backend.midi

class platypush.backend.midi.**MidiBackend**(*device_name=None*, *port_number=None*, *midi_throttle_time=None*, **args*, ***kwargs*)

This backend will listen for events from a MIDI device and post a MidiMessageEvent whenever a new MIDI event happens.

Triggers:

- *platypush.message.event.midi.MidiMessageEvent* when a new MIDI event is received

Requires:

- **rtmidi** (`pip install rtmidi`)

__init__(*device_name=None*, *port_number=None*, *midi_throttle_time=None*, **args*, ***kwargs*)

Parameters

- **device_name** (*str*) – Name of the MIDI device. *N.B.* either *device_name* or *port_number* must be set. Use `platypush.plugins.midi.query_ports()` to get the available ports indices and names
- **port_number** (*int*) – MIDI port number
- **midi_throttle_time** (*int*) – If set, the MIDI events will be throttled - max one per selected time frame (in seconds). Set this parameter if you want to synchronize MIDI events with plugins that normally operate with a lower throughput.

run()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.32 platypush.backend.mqtt

class platypush.backend.mqtt.**MqttBackend**(*host: Optional[str] = None*, *port: int = 1883*, *topic='platypush_bus_mq'*, *subscribe_default_topic: bool = True*, *tls_cafile: Optional[str] = None*, *tls_certfile: Optional[str] = None*, *tls_keyfile: Optional[str] = None*, *tls_version: Optional[str] = None*, *tls_ciphers: Optional[str] = None*, *tls_insecure: bool = False*, *username: Optional[str] = None*, *password: Optional[str] = None*, *client_id: Optional[str] = None*, *listeners=None*, **args*, ***kwargs*)

Backend that reads messages from a configured MQTT topic (default: `platypush_bus_mq/<device_id>`) and posts them to the application bus.

Triggers:

- `platypush.message.event.mqtt.MQTTMessageEvent` **when a new** message is received on one of the custom listeners

Requires:

- **paho-mqtt** (`pip install paho-mqtt`)

`__init__`(*host: Optional[str] = None*, *port: int = 1883*, *topic='platypush_bus_mq'*, *subscribe_default_topic: bool = True*, *tls_cafile: Optional[str] = None*, *tls_certfile: Optional[str] = None*, *tls_keyfile: Optional[str] = None*, *tls_version: Optional[str] = None*, *tls_ciphers: Optional[str] = None*, *tls_insecure: bool = False*, *username: Optional[str] = None*, *password: Optional[str] = None*, *client_id: Optional[str] = None*, *listeners=None*, **args*, ***kwargs*)

Parameters

- **`host`** – MQTT broker host. If no host configuration is specified then the backend will use the host configuration specified on the `mqtt` plugin if it's available.
- **`port`** – MQTT broker port (default: 1883)
- **`topic`** – Topic to read messages from (default: `platypush_bus_mq/<device_id>`)
- **`subscribe_default_topic`** – Whether the backend should subscribe the default topic (default: `platypush_bus_mq/<device_id>`) and execute the messages received there as action requests (default: True).
- **`tls_cafile`** – If TLS/SSL is enabled on the MQTT server and the certificate requires a certificate authority to authenticate it, *ssl_cafile* will point to the provided ca.crt file (default: None)
- **`tls_certfile`** – If TLS/SSL is enabled on the MQTT server and a client certificate it required, specify it here (default: None)
- **`tls_keyfile`** – If TLS/SSL is enabled on the MQTT server and a client certificate key it required, specify it here (default: None) :type tls_keyfile: str
- **`tls_version`** – If TLS/SSL is enabled on the MQTT server and it requires a certain TLS version, specify it here (default: None). Supported versions: `tls` (automatic), `tlsv1`, `tlsv1.1`, `tlsv1.2`.
- **`tls_ciphers`** – If TLS/SSL is enabled on the MQTT server and an explicit list of supported ciphers is required, specify it here (default: None)
- **`tls_insecure`** – Set to True to ignore TLS insecure warnings (default: False).
- **`username`** – Specify it if the MQTT server requires authentication (default: None)
- **`password`** – Specify it if the MQTT server requires authentication (default: None)
- **`client_id`** – ID used to identify the client on the MQTT server (default: None). If None is specified then `Config.get('device_id')` will be used.
- **`listeners`** – If specified then the MQTT backend will also listen for messages on the additional configured message queues. This parameter is a list of maps where each item supports the same arguments passed to the main backend configuration (host, port, topic, password etc.). Note that the message queue configured on the main configuration will expect valid Platypush messages that then can execute, while message queues registered to the listeners will accept any message. Example:

```
listeners:
    - host: localhost
      topics:
          - topic1
          - topic2
          - topic3
    - host: sensors
      topics:
          - topic4
          - topic5
```

on_stop()
Callback invoked when the process stops

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

send_message(*msg*, *topic: Optional[str] = None*, ***kwargs*)
Sends a platypush.message.Message to a node. To be implemented in the derived classes. By default, if the Redis backend is configured then it will try to deliver the message to other consumers through the configured Redis main queue.

Parameters

- **msg** – The message to send
- **queue_name** – Send the message on a specific queue (default: the queue_name configured on the Redis backend)

class platypush.backend.mqtt.**MqttClient**(**args*, *host: str*, *port: int*, *topics: Optional[List[str]] = None*, *on_message: Optional[Callable] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *client_id: Optional[str] = None*, *tls_cafile: Optional[str] = None*, *tls_certfile: Optional[str] = None*, *tls_keyfile: Optional[str] = None*, *tls_version: Optional = None*, *tls_ciphers: Optional = None*, *tls_insecure: bool = False*, *keepalive: Optional[int] = 60*, ***kwargs*)

__init__(**args*, *host: str*, *port: int*, *topics: Optional[List[str]] = None*, *on_message: Optional[Callable] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *client_id: Optional[str] = None*, *tls_cafile: Optional[str] = None*, *tls_certfile: Optional[str] = None*, *tls_keyfile: Optional[str] = None*, *tls_version: Optional = None*, *tls_ciphers: Optional = None*, *tls_insecure: bool = False*, *keepalive: Optional[int] = 60*, ***kwargs*)
client_id is the unique client id string used when connecting to the broker. If client_id is zero length or None, then the behaviour is defined by which protocol version is in use. If using MQTT v3.1.1, then a zero length client id will be sent to the broker and the broker will generate a random for the client. If using MQTT v3.1 then an id will be randomly generated. In both cases, clean_session must be True. If this is not the case a ValueError will be raised.

clean_session is a boolean that determines the client type. If True, the broker will remove all information about this client when it disconnects. If False, the client is a persistent client and subscription information and queued messages will be retained when the client disconnects. Note that a client will never discard its own outgoing messages on disconnect. Calling connect() or reconnect() will cause the messages to be resent. Use reinitialise() to reset a client to its original state. The clean_session argument only applies to MQTT versions v3.1.1 and v3.1. It is not accepted if the MQTT version is v5.0 - use the clean_start argument on connect() instead.

userdata is user defined data of any type that is passed as the "userdata" parameter to callbacks. It may be updated at a later point with the user_data_set() function.

The protocol argument allows explicit setting of the MQTT version to use for this client. Can be paho.mqtt.client.MQTTv311 (v3.1.1), paho.mqtt.client.MQTTv31 (v3.1) or paho.mqtt.client.MQTTv5 (v5.0), with the default being v3.1.1.

Set transport to "websockets" to use WebSockets as the transport mechanism. Set to "tcp" to use raw TCP, which is the default.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

subscribe(**topics*, ***kwargs*)

Subscribe the client to one or more topics.

This function may be called in three different ways (and a further three for MQTT v5.0):

e.g. subscribe("my/topic", 2)

topic: A string specifying the subscription topic to subscribe to. qos: The desired quality of service level for the subscription.

Defaults to 0.

options and properties: Not used.

e.g. subscribe("my/topic", options=SubscribeOptions(qos=2))

topic: A string specifying the subscription topic to subscribe to. qos: Not used. options: The MQTT v5.0 subscribe options. properties: a Properties instance setting the MQTT v5.0 properties to be included. Optional - if not set, no properties are sent.

e.g. subscribe(("my/topic", 1))

topic: A tuple of (topic, qos). Both topic and qos must be present in the tuple.

qos and options: Not used. properties: Only used for MQTT v5.0. A Properties instance setting the MQTT v5.0 properties. Optional - if not set, no properties are sent.

e.g. subscribe(("my/topic", SubscribeOptions(qos=1)))

topic: A tuple of (topic, SubscribeOptions). Both topic and subscribe options must be present in the tuple.

qos and options: Not used. properties: a Properties instance setting the MQTT v5.0 properties to be included. Optional - if not set, no properties are sent.

e.g. subscribe([("my/topic", 0), ("another/topic", 2)])

This allows multiple topic subscriptions in a single SUBSCRIPTION command, which is more efficient than using multiple calls to subscribe().

topic: A list of tuple of format (topic, qos). Both topic and qos must be present in all of the tuples.

qos, options and properties: Not used.

e.g. subscribe([("my/topic", SubscribeOptions(qos=0), ("another/topic", SubscribeOptions(qos=2)])

This allows multiple topic subscriptions in a single SUBSCRIPTION command, which is more efficient than using multiple calls to subscribe().

topic: A list of tuple of format (topic, SubscribeOptions). Both topic and subscribe options must be present in all of the tuples.

qos and options: Not used. properties: a Properties instance setting the MQTT v5.0 properties to be included. Optional - if not set, no properties are sent.

The function returns a tuple (result, mid), where result is MQTT_ERR_SUCCESS to indicate success or (MQTT_ERR_NO_CONN, None) if the client is not currently connected. mid is the message ID for the subscribe request. The mid value can be used to track the subscribe request by checking against the mid argument in the on_subscribe() callback if it is defined.

Raises a ValueError if qos is not 0, 1 or 2, or if topic is None or has zero string length, or if topic is not a string, tuple or list.

`unsubscribe`(**topics*, ***kwargs*)

Unsubscribe the client from one or more topics.

topic: A single string, or list of strings that are the subscription topics to unsubscribe from.

properties: (MQTT v5.0 only) a Properties instance setting the MQTT v5.0 properties to be included. Optional - if not set, no properties are sent.

Returns a tuple (result, mid), where result is MQTT_ERR_SUCCESS to indicate success or (MQTT_ERR_NO_CONN, None) if the client is not currently connected. mid is the message ID for the unsubscribe request. The mid value can be used to track the unsubscribe request by checking against the mid argument in the on_unsubscribe() callback if it is defined.

Raises a ValueError if topic is None or has zero string length, or is not a string or list.

1.33 `platypush.backend.music.mopidy`

class `platypush.backend.music.mopidy.`**`MusicMopidyBackend`**(*host='localhost'*, *port=6680*, ***kwargs*)

This backend listens for events on a Mopidy music server streaming port. Since this backend leverages the Mopidy websocket interface it is only compatible with Mopidy and not with other MPD servers. Please use the *`platypush.backend.music.mpd.MusicMpdBackend`* for a similar polling solution if you're not running Mopidy or your instance has the websocket interface or web port disabled.

Triggers:

- *`platypush.message.event.music.MusicPlayEvent`* if the playback state changed to play
- *`platypush.message.event.music.MusicPauseEvent`* if the playback state changed to pause
- *`platypush.message.event.music.MusicStopEvent`* if the playback state changed to stop
- *`platypush.message.event.music.NewPlayingTrackEvent`* if a new track is being played
- *`platypush.message.event.music.PlaylistChangeEvent`* if the main playlist has changed
- *`platypush.message.event.music.VolumeChangeEvent`* if the main volume has changed
- *`platypush.message.event.music.MuteChangeEvent`* if the mute status has changed
- *`platypush.message.event.music.SeekChangeEvent`* if a track seek event occurs

Requires:

- **websocket-client** (`pip install websocket-client`)

- Mopidy installed and the HTTP service enabled

`__init__`(*host='localhost'*, *port=6680*, ***kwargs*)

Parameters

- **`bus`** – Reference to the bus object to be used in the backend
- **`poll_seconds`** – If the backend implements a `loop` method, this parameter expresses how often the loop should run in seconds.
- **`kwargs`** – Key-value configuration for the backend

`on_stop`()
Callback invoked when the process stops

`run`()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.34 `platypush.backend.music.mpd`

class `platypush.backend.music.mpd.`**`MusicMpdBackend`**(*server='localhost'*, *port=6600*, *poll_seconds=3*, ***kwargs*)

This backend listens for events on a MPD/Mopidy music server.

Triggers:

- *`platypush.message.event.music.MusicPlayEvent`* if the playback state changed to play
- *`platypush.message.event.music.MusicPauseEvent`* if the playback state changed to pause
- *`platypush.message.event.music.MusicStopEvent`* if the playback state changed to stop
- *`platypush.message.event.music.NewPlayingTrackEvent`* if a new track is being played
- *`platypush.message.event.music.PlaylistChangeEvent`* if the main playlist has changed
- *`platypush.message.event.music.VolumeChangeEvent`* if the main volume has changed

Requires:

- **python-mpd2** (`pip install python-mpd2`)
- The *`platypush.plugins.music.mpd`* plugin to be configured

`__init__`(*server='localhost'*, *port=6600*, *poll_seconds=3*, ***kwargs*)

Parameters **`poll_seconds`** (*`float`*) – Interval between queries to the server (default: 3 seconds)

`run`()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.35 `platypush.backend.music.snapcast`

class `platypush.backend.music.snapcast.`**`MusicSnapcastBackend`**(*hosts=None*, *ports=None*, *poll_seconds=10*, **args*, ***kwargs*)

Backend that listens for notification and status changes on one or more [Snapcast](https://github.com/badaix/

snapcast) servers.

Triggers:

- *platypush.message.event.music.snapcast.ClientConnectedEvent*
- *platypush.message.event.music.snapcast.ClientDisconnectedEvent*
- *platypush.message.event.music.snapcast.ClientVolumeChangeEvent*
- *platypush.message.event.music.snapcast.ClientLatencyChangeEvent*
- *platypush.message.event.music.snapcast.ClientNameChangeEvent*
- *platypush.message.event.music.snapcast.GroupMuteChangeEvent*
- *platypush.message.event.music.snapcast.GroupStreamChangeEvent*
- *platypush.message.event.music.snapcast.StreamUpdateEvent*
- *platypush.message.event.music.snapcast.ServerUpdateEvent*

__init__(*hosts=None*, *ports=None*, *poll_seconds=10*, **args*, ***kwargs*)

Parameters

- **hosts** (*list[str]*) – List of Snapcast server names or IPs to monitor (default: *['localhost']*)
- **ports** (*list[int]*) – List of control ports for the configured Snapcast servers (default: *[1705]*)
- **poll_seconds** (*float*) – How often the backend will poll remote servers for status updated (default: 10 seconds)

on_stop()

Callback invoked when the process stops

run()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.36 platypush.backend.nextcloud

class platypush.backend.nextcloud.**NextcloudBackend**(*url: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *object_type: Optional[str] = None*, *object_id: Optional[int] = None*, *poll_seconds: Optional[float] = 60.0*, ***kwargs*)

This backend triggers events when new activities occur on a NextCloud instance.

Triggers:

- *platypush.message.event.nextcloud.NextCloudActivityEvent* when new activity occurs on the instance. The field `activity_type` identifies the activity type (e.g. `file_created`, `file_deleted`, `file_changed`). Example in the case of the creation of new files:

```
{
  "activity_id": 387,
  "app": "files",
  "activity_type": "file_created",
```

(continues on next page)

(continued from previous page)

```
  "user": "your-user",
  "subject": "You created InstantUpload/Camera/IMG_0100.jpg, InstantUpload/
↪Camera/IMG_0101.jpg and InstantUpload/Camera/IMG_0102.jpg",
  "subject_rich": [
    "You created {file3}, {file2} and {file1}",
    {
      "file1": {
        "type": "file",
        "id": "41994",
        "name": "IMG_0100.jpg",
        "path": "InstantUpload/Camera/IMG_0100.jpg",
        "link": "https://your-domain/nextcloud/index.php/f/41994"
      },
      "file2": {
        "type": "file",
        "id": "42005",
        "name": "IMG_0101.jpg",
        "path": "InstantUpload/Camera/IMG_0102.jpg",
        "link": "https://your-domain/nextcloud/index.php/f/42005"
      },
      "file3": {
        "type": "file",
        "id": "42014",
        "name": "IMG_0102.jpg",
        "path": "InstantUpload/Camera/IMG_0102.jpg",
        "link": "https://your-domain/nextcloud/index.php/f/42014"
      }
    }
  ],
  "message": "",
  "message_rich": [
    "",
    []
  ],
  "object_type": "files",
  "object_id": 41994,
  "object_name": "/InstantUpload/Camera/IMG_0102.jpg",
  "objects": {
    "42014": "/InstantUpload/Camera/IMG_0100.jpg",
    "42005": "/InstantUpload/Camera/IMG_0101.jpg",
    "41994": "/InstantUpload/Camera/IMG_0102.jpg"
  },
  "link": "https://your-domain/nextcloud/index.php/apps/files/?dir=/
↪InstantUpload/Camera",
  "icon": "https://your-domain/nextcloud/apps/files/img/add-color.svg",
  "datetime": "2020-09-07T17:04:29+00:00"
}
```

`__init__`(*url: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *object_type: Optional[str] = None*, *object_id: Optional[int] = None*, *poll_seconds: Optional[float] = 60.0*, ***kwargs*)

Parameters

- **url** – NextCloud instance URL (default: same as the `platypush.plugins.nextcloud.NextCloudPlugin`).
- **username** – NextCloud username (default: same as the `platypush.plugins.`

`nextcloud.NextCloudPlugin`).

- **`password`** – NextCloud password (default: same as the `platypush.plugins.nextcloud.NextCloudPlugin`).
- **`object_type`** – If set, only filter events on this type of object.
- **`object_id`** – If set, only filter events on this object ID.
- **`poll_seconds`** – How often the backend should poll the instance (default: one minute).

1.37 `platypush.backend.nfc`

class `platypush.backend.nfc.`**`NfcBackend`**(*device='usb'*, **args*, ***kwargs*)

Backend to detect NFC card events from a compatible reader.

Triggers:

- *`platypush.message.event.nfc.NFCDeviceConnectedEvent`* when an NFC reader/writer is connected
- *`platypush.message.event.nfc.NFCDeviceDisconnectedEvent`* when an NFC reader/writer is disconnected
- *`platypush.message.event.nfc.NFCTagDetectedEvent`* when an NFC tag is detected
- *`platypush.message.event.nfc.NFCTagRemovedEvent`* when an NFC tag is removed

Requires:

- **nfcpy** >= 1.0 (`pip install 'nfcpy>=1.0'`)
- **ndef** (`pip install ndef`)

Run the following to check if your device is compatible with nfcpy and the right permissions are set:

```
python -m nfc
```

`__init__`(*device='usb'*, **args*, ***kwargs*)

Parameters **`device`** – Address or ID of the device to be opened. Examples:

- *'usb:003:009'* opens device 9 on bus 3
- *'usb:003'* opens the first available device on bus 3
- *'usb'* opens the first available USB device (default)

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.38 `platypush.backend.nodered`

class `platypush.backend.nodered.`**`NoderedBackend`**(*port: int = 5051*, **args*, ***kwargs*)

This backend publishes platypush actions to a Node-RED instance. If you enable this backend on a host that runs Node-RED then a new block (platypush -> run) can be used in your flows. This block will accept JSON requests as input in the format `{"type":"request", "action":"plugin.name.action_name", "args": {...}}` and return the output of the action as block output, or raise an exception if the action failed.

Requires:

- **pynodered** (`pip install pynodered`)

`__init__`(*port: int = 5051*, **args*, ***kwargs*)

Parameters **`port`** – Listening port for the local publishing web server (default: 5051)

`on_stop`()
Callback invoked when the process stops

`run`()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.39 `platypush.backend.ping`

class `platypush.backend.ping.`**`PingBackend`**(*hosts: List[str], timeout: float = 5.0, interval: float = 60.0, count: int = 1, *args, **kwargs*)
This backend allows you to ping multiple remote hosts at regular intervals.

Triggers:

- *`platypush.message.event.ping.HostDownEvent`* if a host stops responding ping requests
- *`platypush.message.event.ping.HostUpEvent`* if a host starts responding ping requests

class **`Pinger`**(**args*, ***kwargs*)

`__init__`(**args*, ***kwargs*)
Parameters
- **`request_queue`** – The worker will listen for messages to process over this queue
- **`response_queue`** – The worker will return responses over this queue

`process`(*host: str*) → Tuple[str, bool]
This method must be implemented by the derived classes. It will take as argument a message received over the *request_queue* and will return a value that will be processed by the consumer or None.

If this function raises an exception then the exception will be pushed to the response queue and can be handled by the consumer.

`__init__`(*hosts: List[str], timeout: float = 5.0, interval: float = 60.0, count: int = 1, *args, **kwargs*)

Parameters

- **`hosts`** – List of IP addresses or host names to monitor.
- **`timeout`** – Ping timeout.
- **`interval`** – Interval between two scans.
- **`count`** – Number of pings per host. A host will be considered down if all the ping requests fail.

`run`()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.40 `platypush.backend.pushbullet`

class `platypush.backend.pushbullet.`**`PushbulletBackend`**(*token: str, device: str = 'Platypush', proxy_host: Optional[str] = None, proxy_port: Optional[int] = None, **kwargs*)

This backend will listen for events on a Pushbullet (https://pushbullet.com) channel and propagate them to the bus. This backend is quite useful if you want to synchronize events and actions with your mobile phone (through the Pushbullet app and/or through Tasker), synchronize clipboards, send pictures and files to other devices etc. You can also wrap Platypush messages as JSON into a push body to execute them.

Triggers:

- *`platypush.message.event.pushbullet.PushbulletEvent`* if a new push is received

Requires:

- **requests** (`pip install requests`)
- **pushbullet.py** (`pip install git+https://github.com/rbrcsk/pushbullet.py`)

`__init__`(*token: str, device: str = 'Platypush', proxy_host: Optional[str] = None, proxy_port: Optional[int] = None, **kwargs*)

Parameters

- **`token`** – Your Pushbullet API token, see https://docs.pushbullet.com/#authentication
- **`device`** – Name of the virtual device for Platypush (default: Platypush)
- **`proxy_host`** – HTTP proxy host (default: None)
- **`proxy_port`** – HTTP proxy port (default: None)

`on_stop`()

Callback invoked when the process stops

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.41 `platypush.backend.redis`

class `platypush.backend.redis.`**`RedisBackend`**(*queue='platypush_bus_mq', redis_args=None, *args, **kwargs*)

Backend that reads messages from a configured Redis queue (default: `platypush_bus_mq`) and posts them to the application bus. Very useful when you have plugin whose code is executed in another process and can't post events or requests to the application bus.

Requires:

- **redis** (`pip install redis`)

`__init__`(*queue='platypush_bus_mq', redis_args=None, *args, **kwargs*)

Parameters

- **`queue`** (*`str`*) – Queue name to listen on (default: `platypush_bus_mq`)
- **`redis_args`** (*`dict`*) – Arguments that will be passed to the redis-py constructor (e.g. host, port, password), see http://redis-py.readthedocs.io/en/latest/

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

send_message(*msg*, *queue_name=None*, ***kwargs*)
Sends a platypush.message.Message to a node. To be implemented in the derived classes. By default, if the Redis backend is configured then it will try to deliver the message to other consumers through the configured Redis main queue.

Parameters

- **msg** – The message to send
- **queue_name** – Send the message on a specific queue (default: the queue_name configured on the Redis backend)

1.42 platypush.backend.scard

class platypush.backend.scard.**ScardBackend**(*atr=None*, **args*, ***kwargs*)
Generic backend to read smart cards and NFC tags and trigger an event whenever a device is detected.

Extend this backend to implement more advanced communication with custom smart cards.

Triggers:

- *platypush.message.event.scard.SmartCardDetectedEvent* when a smart card is detected
- *platypush.message.event.scard.SmartCardRemovedEvent* when a smart card is removed

Requires:

- **pyscard** (`pip install pyscard`)

__init__(*atr=None*, **args*, ***kwargs*)

Parameters **atr** – If set, the backend will trigger events only for card(s) with the specified ATR(s). It can be either an ATR string (space-separated hex octects) or a list of ATR strings. Default: none (any card will be detected)

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.43 platypush.backend.sensor

class platypush.backend.sensor.**SensorBackend**(*plugin=None*, *plugin_args=None*, *thresholds=None*, *tolerance=1e-07*, *poll_seconds=None*, *enabled_sensors=None*, ***kwargs*)
Abstract backend for polling sensors.

Triggers:

- *platypush.message.event.sensor.SensorDataChangeEvent* if the measurements of a sensor have changed
- ***platypush.message.event.sensor.SensorDataAboveThresholdEvent*** **if the measurements of a sensor**
  gone above a configured threshold

- `platypush.message.event.sensor.SensorDataBelowThresholdEvent` **if the measurements of a sensor** gone below a configured threshold

`__init__`(*plugin=None*, *plugin_args=None*, *thresholds=None*, *tolerance=1e-07*, *poll_seconds=None*, *enabled_sensors=None*, ***kwargs*)

Parameters

- **plugin** (*str*) – If set, then this plugin instance, referenced by plugin id, will be polled through `get_plugin()`. Example: `'gpio.sensor.bme280'` or `'gpio.sensor.envirophat'`.
- **plugin_args** (*dict*) – If plugin is set and its `get_measurement()` method accepts optional arguments, then you can pass those arguments through `plugin_args`.
- **thresholds** – Thresholds can be either a scalar value or a dictionary (e.g. `{"temperature": 20.0}`). Sensor threshold events will be fired when measurements get above or below these values. Set it as a scalar if your get_measurement() code returns a scalar, as a dictionary if it returns a dictionary of values. For instance, if your sensor code returns both humidity and temperature in a format like `{'humidity':60.0, 'temperature': 25.0}`, you'll want to set up a threshold on temperature with a syntax like `{'temperature':20.0}` to trigger events when the temperature goes above/below 20 degrees.
- **tolerance** (*dict or float*) – If set, then the sensor change events will be triggered only if the difference between the new value and the previous value is higher than the specified tolerance. Example:

```
{
    "temperature": 0.01,  # Tolerance on the 2nd decimal digit
    "humidity": 0.1       # Tolerance on the 1st decimal digit
}
```

- **poll_seconds** (*float*) – If set, the thread will wait for the specified number of seconds between a read and the next one.
- **enabled_sensors** (dict (in the form `name -> [True/False]`), set or list) – If `get_measurement()` returns data in dict form, then `enabled_sensors` selects which keys should be taken into account when monitoring for new events (e.g. "temperature" or "humidity").

`get_measurement`()
Wrapper around `plugin.get_measurement()` that can filter events on specified enabled sensors data or on specified tolerance values. It can be overridden by derived classes.

`on_stop`()
Callback invoked when the process stops

`run`()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.44 `platypush.backend.sensor.accelerometer`

class `platypush.backend.sensor.accelerometer.`**`SensorAccelerometerBackend`**(***kwargs*)
Backend to poll position information from an accelerometer sensor.

Requires:

- `Adafruit_Python_GPIO` (`pip install Adafruit_Python_GPIO`)

- The *platypush.plugins.gpio.sensor.accelerometer* plugin configured

__init__(***kwargs*)

Parameters

- **plugin** (*str*) – If set, then this plugin instance, referenced by plugin id, will be polled through `get_plugin()`. Example: `'gpio.sensor.bme280'` or `'gpio.sensor.envirophat'`.
- **plugin_args** (*dict*) – If plugin is set and its `get_measurement()` method accepts optional arguments, then you can pass those arguments through `plugin_args`.
- **thresholds** – Thresholds can be either a scalar value or a dictionary (e.g. `{"temperature": 20.0}`). Sensor threshold events will be fired when measurements get above or below these values. Set it as a scalar if your get_measurement() code returns a scalar, as a dictionary if it returns a dictionary of values. For instance, if your sensor code returns both humidity and temperature in a format like `{'humidity':60.0, 'temperature': 25.0}`, you'll want to set up a threshold on temperature with a syntax like `{'temperature':20.0}` to trigger events when the temperature goes above/below 20 degrees.
- **tolerance** (*dict or float*) – If set, then the sensor change events will be triggered only if the difference between the new value and the previous value is higher than the specified tolerance. Example:

```
{
    "temperature": 0.01,  # Tolerance on the 2nd decimal digit
    "humidity": 0.1       # Tolerance on the 1st decimal digit
}
```

- **poll_seconds** (*float*) – If set, the thread will wait for the specified number of seconds between a read and the next one.
- **enabled_sensors** (dict (in the form `name -> [True/False]`), set or list) – If `get_measurement()` returns data in dict form, then `enabled_sensors` selects which keys should be taken into account when monitoring for new events (e.g. "temperature" or "humidity").

1.45 platypush.backend.sensor.arduino

class platypush.backend.sensor.arduino.**SensorArduinoBackend**(***kwargs*)

This backend listens for new events from an Arduino with a Firmata-compatible firmware.

Requires:

- The *platypush.plugins.arduino.ArduinoPlugin* plugin configured.

__init__(***kwargs*)

Parameters

- **plugin** (*str*) – If set, then this plugin instance, referenced by plugin id, will be polled through `get_plugin()`. Example: `'gpio.sensor.bme280'` or `'gpio.sensor.envirophat'`.
- **plugin_args** (*dict*) – If plugin is set and its `get_measurement()` method accepts optional arguments, then you can pass those arguments through `plugin_args`.

- **thresholds** – Thresholds can be either a scalar value or a dictionary (e.g. `{"temperature": 20.0}`). Sensor threshold events will be fired when measurements get above or below these values. Set it as a scalar if your get_measurement() code returns a scalar, as a dictionary if it returns a dictionary of values. For instance, if your sensor code returns both humidity and temperature in a format like `{'humidity':60.0, 'temperature': 25.0}`, you'll want to set up a threshold on temperature with a syntax like `{'temperature':20.0}` to trigger events when the temperature goes above/below 20 degrees.
- **tolerance** (*dict or float*) – If set, then the sensor change events will be triggered only if the difference between the new value and the previous value is higher than the specified tolerance. Example:

```
{
    "temperature": 0.01,  # Tolerance on the 2nd decimal digit
    "humidity": 0.1       # Tolerance on the 1st decimal digit
}
```

- **poll_seconds** (*float*) – If set, the thread will wait for the specified number of seconds between a read and the next one.
- **enabled_sensors** (dict (in the form `name -> [True/False]`), set or list) – If `get_measurement()` returns data in dict form, then `enabled_sensors` selects which keys should be taken into account when monitoring for new events (e.g. "temperature" or "humidity").

1.46 platypush.backend.sensor.battery

class platypush.backend.sensor.battery.**SensorBatteryBackend**(***kwargs*)

This backend listens for battery full/connect/disconnect/below/above threshold events. The sensor events triggered by this backend will include any of the following fields:

- `battery_percent`
- `battery_power_plugged`

Requires:

- **psutil** (`pip install psutil`) for CPU load and stats.

__init__(***kwargs*)

Parameters

- **plugin** (*str*) – If set, then this plugin instance, referenced by plugin id, will be polled through `get_plugin()`. Example: `'gpio.sensor.bme280'` or `'gpio.sensor.envirophat'`.
- **plugin_args** (*dict*) – If plugin is set and its `get_measurement()` method accepts optional arguments, then you can pass those arguments through `plugin_args`.
- **thresholds** – Thresholds can be either a scalar value or a dictionary (e.g. `{"temperature": 20.0}`). Sensor threshold events will be fired when measurements get above or below these values. Set it as a scalar if your get_measurement() code returns a scalar, as a dictionary if it returns a dictionary of values. For instance, if your sensor code returns both humidity and temperature in a format like `{'humidity':60.0, 'temperature': 25.0}`, you'll want to set up a threshold on temperature with

a syntax like `{'temperature':20.0}` to trigger events when the temperature goes above/below 20 degrees.

- **`tolerance`** (*dict or float*) – If set, then the sensor change events will be triggered only if the difference between the new value and the previous value is higher than the specified tolerance. Example:

```
{
    "temperature": 0.01,   # Tolerance on the 2nd decimal digit
    "humidity": 0.1        # Tolerance on the 1st decimal digit
}
```

- **`poll_seconds`** (*float*) – If set, the thread will wait for the specified number of seconds between a read and the next one.
- **`enabled_sensors`** (dict (in the form `name -> [True/False]`), set or list) – If `get_measurement()` returns data in dict form, then `enabled_sensors` selects which keys should be taken into account when monitoring for new events (e.g. "temperature" or "humidity").

`get_measurement``()`
Wrapper around `plugin.get_measurement()` that can filter events on specified enabled sensors data or on specified tolerance values. It can be overridden by derived classes.

1.47 `platypush.backend.sensor.bme280`

class `platypush.backend.sensor.bme280.`**`SensorBme280Backend`**(*temperature=True, pressure=True, humidity=True, **kwargs*)

Backend to poll analog sensor values from a BME280 environment sensor

Requires:

- `pimoroni-bme280` (`pip install pimoroni-bme280`)

`__init__`(*temperature=True, pressure=True, humidity=True, **kwargs*)

Parameters

- **`temperature`** – Enable temperature sensor polling
- **`pressure`** – Enable pressure sensor polling
- **`humidity`** – Enable humidity sensor polling

1.48 `platypush.backend.sensor.dht`

class `platypush.backend.sensor.dht.`**`SensorDhtBackend`**(*temperature: bool = True, humidity: bool = True, **kwargs*)

Backend to poll a DHT11/DHT22/AM2302 temperature/humidity sensor.

Requires:

- `Adafruit_Python_DHT` (`pip install git+https://github.com/adafruit/Adafruit_Python_DHT.git`)
- The `gpio.sensor.dht` plugin configured and enabled.

`__init__`(*temperature: bool = True, humidity: bool = True, **kwargs*)

Parameters

- **temperature** – Enable temperature sensor poll.
- **humidity** – Enable humidity sensor poll.

1.49 platypush.backend.sensor.distance

class platypush.backend.sensor.distance.**SensorDistanceBackend**(*plugin=None*, *plugin_args=None*, *thresholds=None*, *tolerance=1e-07*, *poll_seconds=None*, *enabled_sensors=None*, ***kwargs*)

Backend to poll a distance sensor.

Requires:

- `RPi.GPIO` (`pip install RPi.GPIO`)
- The *platypush.plugins.gpio.sensor.distance* plugin configured

get_measurement()
get_measurement implementation

1.50 platypush.backend.sensor.distance.vl53l1x

class platypush.backend.sensor.distance.vl53l1x.**SensorDistanceVl53L1XBackend**(*short=True*, *medium=False*, *long=False*, ***kwargs*)

Backend to poll an VL53L1x laser ranger/distance sensor

Requires:

- `smbus2` (`pip install smbus2`)
- `vl53l1x` (`pip install vl53l1x`)

__init__(*short=True*, *medium=False*, *long=False*, ***kwargs*)

Parameters

- **short** – Enable short range measurement (default: True)
- **medium** – Enable medium range measurement (default: False)
- **long** – Enable long range measurement (default: False)

1.51 platypush.backend.sensor.envirophat

class platypush.backend.sensor.envirophat.**SensorEnvirophatBackend**(*temperature=True, pressure=True, altitude=True, luminosity=True, analog=True, accelerometer=True, magnetometer=True, qnh=1020, **kwargs*)

Backend to poll analog sensor values from an enviroPHAT sensor pHAT (https://shop.pimoroni.com/products/enviro-phat)

Requires:

- `envirophat` (`pip install envirophat`)

__init__(*temperature=True, pressure=True, altitude=True, luminosity=True, analog=True, accelerometer=True, magnetometer=True, qnh=1020, **kwargs*)

Parameters

- **temperature** – Enable temperature sensor polling
- **pressure** – Enable pressure sensor polling
- **altitude** – Enable altitude sensor polling
- **luminosity** – Enable luminosity sensor polling
- **analog** – Enable analog sensors polling
- **accelerometer** – Enable accelerometer polling
- **magnetometer** – Enable magnetometer polling
- **qnh** – Base reference for your sea level pressure (for altitude sensor)

1.52 platypush.backend.sensor.ir.zeroborg

class platypush.backend.sensor.ir.zeroborg.**SensorIrZeroborgBackend**(*no_message_timeout=0.37, **kwargs*)

This backend will read for events on the infrared sensor of a ZeroBorg (https://www.piborg.org/motor-control-1135/zeroborg) circuitry for Raspberry Pi. You can see the codes associated to an IR event from any remote by running the scan utility:

```
python -m platypush.backend.sensor.ir.zeroborg.scan
```

Triggers:

- *platypush.message.event.sensor.ir.IrKeyDownEvent* when a key is pressed

- `platypush.message.event.sensor.ir.IrKeyUpEvent` when a key is released

`__init__`(*no_message_timeout=0.37*, ***kwargs*)

Parameters

- **`bus`** – Reference to the bus object to be used in the backend
- **`poll_seconds`** – If the backend implements a `loop` method, this parameter expresses how often the loop should run in seconds.
- **`kwargs`** – Key-value configuration for the backend

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.53 `platypush.backend.sensor.leap`

class `platypush.backend.sensor.leap.`**`LeapFuture`**(*seconds*, *listener*, *event*)

`__init__`(*seconds*, *listener*, *event*)

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

class `platypush.backend.sensor.leap.`**`LeapListener`**(*position_ranges*, *position_tolerance*, *logger*, *frames_throttle_secs=None*)

`__init__`(*position_ranges*, *position_tolerance*, *logger*, *frames_throttle_secs=None*)

Initialize self. See help(type(self)) for accurate signature.

class `platypush.backend.sensor.leap.`**`SensorLeapBackend`**(*position_ranges=None*, *position_tolerance=0.0*, *frames_throttle_secs=None*, **args*, ***kwargs*)

Backend for events generated using a Leap Motion device to track hands and gestures, https://www.leapmotion.com/

Note that the default SDK is not compatible with Python 3. Follow the instructions on https://github.com/BlackLight/leap-sdk-python3 to build the Python 3 module.

Also, you'll need the Leap driver and utils installed on your OS (follow instructions at https://www.leapmotion.com/setup/) and the *leapd* daemon running to recognize your controller.

Requires:

- The Redis backend enabled

- The Leap Motion SDK compiled with Python 3 support, see my port at https://github.com:BlackLight/leap-sdk-python3.git
- The *leapd* daemon to be running and your Leap Motion connected

Triggers:

- `platypush.message.event.sensor.leap.LeapFrameEvent` when a new frame is received
- `platypush.message.event.sensor.leap.LeapFrameStartEvent` when a new sequence of frame starts
- `platypush.message.event.sensor.leap.LeapFrameStopEvent` when a sequence of frame stops
- `platypush.message.event.sensor.leap.LeapConnectEvent` when a Leap Motion device is connected
- `platypush.message.event.sensor.leap.LeapDisconnectEvent` when a Leap Motion device disconnects

`__init__`(*position_ranges=None*, *position_tolerance=0.0*, *frames_throttle_secs=None*, **args*, ***kwargs*)

Parameters `position_ranges` – It specifies how wide the hand space (x, y and z axes) should be in millimiters.

Default:

```
[
    [-300.0, 300.0],  # x axis
    [25.0, 600.0],    # y axis
    [-300.0, 300.0],  # z axis
]
```

Parameters

- **`position_tolerance`** (*float*) – % of change between a frame and the next to really consider the next frame as a new one (default: 0)
- **`frames_throttle_secs`** (*float*) – If set, the frame events will be throttled and pushed to the main queue at the specified rate. Good to set if you want to connect Leap Motion events to actions that have a lower throughput (the Leap Motion can send a lot of frames per second). Default: None (no throttling)

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.54 `platypush.backend.sensor.ltr559`

class `platypush.backend.sensor.ltr559.`**`SensorLtr559Backend`**(*light=True*, *proximity=True*, ***kwargs*)

Backend to poll an LTR559 light/proximity sensor

Requires:

- `ltr559` (`pip install ltr559`)

`__init__`(*light=True*, *proximity=True*, ***kwargs*)

Parameters

- **light** – Enable light sensor
- **proximity** – Enable proximity sensor

1.55 platypush.backend.sensor.mcp3008

class platypush.backend.sensor.mcp3008.**SensorMcp3008Backend**(***kwargs*)

Backend to poll analog sensor values from an MCP3008 chipset (https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/mcp3008)

Requires:

- `adafruit-mcp3008` (`pip install adafruit-mcp3008`)
- The *platypush.plugins.gpio.sensor.mcp3008* plugin configured

__init__(***kwargs*)

Parameters

- **plugin** (*str*) – If set, then this plugin instance, referenced by plugin id, will be polled through `get_plugin()`. Example: `'gpio.sensor.bme280'` or `'gpio.sensor.envirophat'`.
- **plugin_args** (*dict*) – If plugin is set and its `get_measurement()` method accepts optional arguments, then you can pass those arguments through `plugin_args`.
- **thresholds** – Thresholds can be either a scalar value or a dictionary (e.g. `{"temperature": 20.0}`). Sensor threshold events will be fired when measurements get above or below these values. Set it as a scalar if your get_measurement() code returns a scalar, as a dictionary if it returns a dictionary of values. For instance, if your sensor code returns both humidity and temperature in a format like `{'humidity':60.0, 'temperature': 25.0}`, you'll want to set up a threshold on temperature with a syntax like `{'temperature':20.0}` to trigger events when the temperature goes above/below 20 degrees.
- **tolerance** (*dict or float*) – If set, then the sensor change events will be triggered only if the difference between the new value and the previous value is higher than the specified tolerance. Example:

```
{
    "temperature": 0.01,  # Tolerance on the 2nd decimal digit
    "humidity": 0.1       # Tolerance on the 1st decimal digit
}
```

- **poll_seconds** (*float*) – If set, the thread will wait for the specified number of seconds between a read and the next one.
- **enabled_sensors** (dict (in the form `name -> [True/False]`), set or list) – If `get_measurement()` returns data in dict form, then `enabled_sensors` selects which keys should be taken into account when monitoring for new events (e.g. "temperature" or "humidity").

1.56 platypush.backend.sensor.motion.pwm3901

class platypush.backend.sensor.motion.pwm3901.**SensorMotionPwm3901Backend**(***kwargs*)

Backend to poll an PWM3901 optical flow and motion sensor

Requires:

- pwm3901 (pip install pwm3901)

__init__(***kwargs*)

Parameters

- **plugin** (*str*) – If set, then this plugin instance, referenced by plugin id, will be polled through get_plugin(). Example: 'gpio.sensor.bme280' or 'gpio.sensor.envirophat'.
- **plugin_args** (*dict*) – If plugin is set and its get_measurement() method accepts optional arguments, then you can pass those arguments through plugin_args.
- **thresholds** – Thresholds can be either a scalar value or a dictionary (e.g. {"temperature": 20.0}). Sensor threshold events will be fired when measurements get above or below these values. Set it as a scalar if your get_measurement() code returns a scalar, as a dictionary if it returns a dictionary of values. For instance, if your sensor code returns both humidity and temperature in a format like {'humidity':60.0, 'temperature': 25.0}, you'll want to set up a threshold on temperature with a syntax like {'temperature':20.0} to trigger events when the temperature goes above/below 20 degrees.
- **tolerance** (*dict or float*) – If set, then the sensor change events will be triggered only if the difference between the new value and the previous value is higher than the specified tolerance. Example:

```
{
    "temperature": 0.01,  # Tolerance on the 2nd decimal digit
    "humidity": 0.1       # Tolerance on the 1st decimal digit
}
```

- **poll_seconds** (*float*) – If set, the thread will wait for the specified number of seconds between a read and the next one.
- **enabled_sensors** (dict (in the form name -> [True/False]), set or list) – If get_measurement() returns data in dict form, then enabled_sensors selects which keys should be taken into account when monitoring for new events (e.g. "temperature" or "humidity").

1.57 platypush.backend.sensor.serial

class platypush.backend.sensor.serial.**SensorSerialBackend**(***kwargs*)

This backend listens for new events from sensors connected through a serial interface (like Arduino) acting as a wrapper for the serial plugin.

Requires:

- The *platypush.plugins.serial* plugin configured

__init__(***kwargs*)

Parameters

- **plugin** (*str*) – If set, then this plugin instance, referenced by plugin id, will be polled through get_plugin(). Example: 'gpio.sensor.bme280' or 'gpio.sensor.envirophat'.

- **plugin_args** (*dict*) – If plugin is set and its `get_measurement()` method accepts optional arguments, then you can pass those arguments through `plugin_args`.
- **thresholds** – Thresholds can be either a scalar value or a dictionary (e.g. `{"temperature": 20.0}`). Sensor threshold events will be fired when measurements get above or below these values. Set it as a scalar if your get_measurement() code returns a scalar, as a dictionary if it returns a dictionary of values. For instance, if your sensor code returns both humidity and temperature in a format like `{'humidity':60.0, 'temperature': 25.0}`, you'll want to set up a threshold on temperature with a syntax like `{'temperature':20.0}` to trigger events when the temperature goes above/below 20 degrees.
- **tolerance** (*dict or float*) – If set, then the sensor change events will be triggered only if the difference between the new value and the previous value is higher than the specified tolerance. Example:

```
{
    "temperature": 0.01,   # Tolerance on the 2nd decimal digit
    "humidity": 0.1        # Tolerance on the 1st decimal digit
}
```

- **poll_seconds** (*float*) – If set, the thread will wait for the specified number of seconds between a read and the next one.
- **enabled_sensors** (dict (in the form `name -> [True/False]`), set or list) – If `get_measurement()` returns data in dict form, then `enabled_sensors` selects which keys should be taken into account when monitoring for new events (e.g. "temperature" or "humidity").

1.58 platypush.backend.stt

class platypush.backend.stt.**SttBackend**(*plugin_name: str, retry_sleep: float = 5.0, *args, **kwargs*)

Base class for speech-to-text backends.

__init__(*plugin_name: str, retry_sleep: float = 5.0, *args, **kwargs*)

Parameters

- **plugin_name** – Plugin name of the class that will be used for speech detection. Must be an instance of *platypush.plugins.stt.SttPlugin*.
- **retry_sleep** – Number of seconds the backend will wait on failure before re-initializing the plugin (default: 5 seconds).

run()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.59 platypush.backend.stt.deepspeech

class platypush.backend.stt.deepspeech.**SttDeepspeechBackend**(**args, **kwargs*)

Backend for the Mozilla Deepspeech speech-to-text engine plugin. Set this plugin to `enabled` if you want to run the speech-to-text engine continuously instead of programmatically using `start_detection` and `stop_detection`.

Requires:

- The *platypush.plugins.stt.deepspeech.SttDeepspeechPlugin* plugin configured and its dependencies installed, as well as the language model files.

__init__(**args*, ***kwargs*)

Parameters

- **plugin_name** – Plugin name of the class that will be used for speech detection. Must be an instance of *platypush.plugins.stt.SttPlugin*.
- **retry_sleep** – Number of seconds the backend will wait on failure before re-initializing the plugin (default: 5 seconds).

1.60 platypush.backend.stt.picovoice.hotword

class platypush.backend.stt.picovoice.hotword.**SttPicovoiceHotwordBackend**(**args*, ***kwargs*)

Backend for the PicoVoice hotword detection plugin. Set this plugin to `enabled` if you want to run the hotword engine continuously instead of programmatically using `start_detection` and `stop_detection`.

Requires:

- The `platypush.plugins.stt.deepspeech.SttPicovoiceHotwordPlugin` plugin configured and its dependencies installed.

__init__(**args*, ***kwargs*)

Parameters

- **plugin_name** – Plugin name of the class that will be used for speech detection. Must be an instance of *platypush.plugins.stt.SttPlugin*.
- **retry_sleep** – Number of seconds the backend will wait on failure before re-initializing the plugin (default: 5 seconds).

1.61 platypush.backend.stt.picovoice.speech

class platypush.backend.stt.picovoice.speech.**SttPicovoiceSpeechBackend**(**args*, ***kwargs*)

Backend for the PicoVoice speech detection plugin. Set this plugin to `enabled` if you want to run the speech engine continuously instead of programmatically using `start_detection` and `stop_detection`.

Requires:

- The `platypush.plugins.stt.deepspeech.SttPicovoiceSpeechPlugin` plugin configured and its dependencies installed.

__init__(**args*, ***kwargs*)

Parameters

- **plugin_name** – Plugin name of the class that will be used for speech detection. Must be an instance of *platypush.plugins.stt.SttPlugin*.
- **retry_sleep** – Number of seconds the backend will wait on failure before re-initializing the plugin (default: 5 seconds).

1.62 platypush.backend.tcp

class platypush.backend.tcp.**TcpBackend**(*port*, *bind_address=None*, *listen_queue=5*, **args*, ***kwargs*)

Backend that reads messages from a configured TCP port

__init__(*port*, *bind_address=None*, *listen_queue=5*, **args*, ***kwargs*)

Parameters

- **port** (*int*) – TCP port number
- **bind_address** (*str*) – Specify a bind address if you want to hook the service to a specific interface (default: listen for any connections)
- **listen_queue** (*int*) – Maximum number of queued connections (default: 5)

run()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.63 platypush.backend.todoist

class platypush.backend.todoist.**TodoistBackend**(*api_token: str = None*, ***kwargs*)

This backend listens for events on a remote Todoist account.

Requires:

- **todoist-python** (`pip install todoist-python`)
- **websocket-client** (`pip install websocket-client`)

Triggers:

- *platypush.message.event.todoist.NewItemEvent* when a new item is created.
- *platypush.message.event.todoist.RemovedItemEvent* when an item is removed.
- *platypush.message.event.todoist.CheckedItemEvent* when an item is checked.
- *platypush.message.event.todoist.ItemContentChangeEvent* when the content of an item is changed.
- ***platypush.message.event.todoist.ModifiedItemEvent* when an item is changed and the change**
  doesn't fall into the categories above.
- ***platypush.message.event.todoist.TodoistSyncRequiredEvent* when an update has occurred that do**
  fall into the categories above and a sync is required to get up-to-date.

__init__(*api_token: str = None*, ***kwargs*)

Parameters

- **bus** – Reference to the bus object to be used in the backend
- **poll_seconds** – If the backend implements a `loop` method, this parameter expresses how often the loop should run in seconds.
- **kwargs** – Key-value configuration for the backend

run()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.64 `platypush.backend.travisci`

class platypush.backend.travisci.**TravisciBackend**(*poll_seconds: Optional[float] = 60.0*, **args*, ***kwargs*)

This backend polls for new builds on a Travis-Ci account and triggers an event whenever a new build is completed.

Requires:

- The *platypush.plugins.foursquare.FoursquarePlugin* plugin configured and enabled.

Triggers:

- ***platypush.message.event.travisci.TravisciBuildPassedEvent* when the build of a project owned b**
  the user passes.
- ***platypush.message.event.travisci.TravisciBuildFailedEvent* when the build of a project owned b**
  the user fails.

__init__(*poll_seconds: Optional[float] = 60.0*, **args*, ***kwargs*)

Parameters **poll_seconds** – How often the backend should check for new builds (default: one minute).

1.65 `platypush.backend.trello`

class platypush.backend.trello.**TrelloBackend**(*boards: List[str], token: str, **kwargs*)

This backend listens for events on a remote Trello account through websocket interface.. Note that the Trello websocket interface is not officially supported and it requires a different token from the one you use for the Trello API (and for the Trello plugin). To get the websocket token:

1. Open https://trello.com in your browser.
2. Open the developer tools (F12), go to the Network tab, select 'Websocket' or 'WS' in the filter bar and refresh the page.
3. You should see an entry in the format `wss://trello.com/1/Session/socket?token=<token>`.
4. Copy the `<token>` in the configuration of this backend.

Requires:

- **websocket-client** (`pip install websocket-client`)
- The *platypush.plugins.trello.TrelloPlugin* configured.

Triggers:

- *platypush.message.event.trello.NewCardEvent* when a card is created.
- `platypush.message.event.MoveCardEvent` when a card is moved.
- `platypush.message.event.ArchivedCardEvent` when a card is archived/closed.
- `platypush.message.event.UnarchivedCardEvent` when a card is un-archived/opened.

__init__(*boards: List[str], token: str, **kwargs*)

Parameters

- **boards** – List of boards to subscribe, by ID or name.

- **token** – Trello web client API token.

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.66 platypush.backend.weather

class platypush.backend.weather.**WeatherBackend**(*plugin_name: str*, *poll_seconds: int*, ***kwargs*)
Abstract class for weather update backends.

__init__(*plugin_name: str*, *poll_seconds: int*, ***kwargs*)

Parameters

- **plugin_name** – Name of the weather plugin to be used.
- **poll_seconds** – How often the backend should check for updates, in seconds.

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.67 platypush.backend.weather.buienradar

class platypush.backend.weather.buienradar.**WeatherBuienradarBackend**(*poll_seconds=300*, ***kwargs*)
Buienradar weather forecast backend. Listens for new weather or precipitation updates.

Triggers:

- *platypush.message.event.weather.NewWeatherConditionEvent* when there is a weather condition update

Requires:

- The *platypush.plugins.weather.buienradar* plugin configured

__init__(*poll_seconds=300*, ***kwargs*)

Parameters

- **bus** – Reference to the bus object to be used in the backend
- **poll_seconds** – If the backend implements a `loop` method, this parameter expresses how often the loop should run in seconds.
- **kwargs** – Key-value configuration for the backend

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.68 platypush.backend.weather.darksky

class platypush.backend.weather.darksky.**WeatherDarkskyBackend**(*poll_seconds: int = 300*, ***kwargs*)
Weather forecast backend that leverages the DarkSky API.

Triggers:

- `platypush.message.event.weather.NewWeatherConditionEvent` when there is a weather condition update

Requires:

- The `platypush.plugins.weather.darksky.WeatherDarkskyPlugin` plugin configured

`__init__`(*poll_seconds: int = 300*, ***kwargs*)

Parameters **`poll_seconds`** – How often the backend should check for updates (default: every 5 minutes).

1.69 `platypush.backend.weather.openweathermap`

class `platypush.backend.weather.openweathermap.`**`WeatherOpenweathermapBackend`**(*poll_seconds: int = 60*, ***kwargs*)

Weather forecast backend that leverages the OpenWeatherMap API.

Triggers:

- `platypush.message.event.weather.NewWeatherConditionEvent` when there is a weather condition update

Requires:

- The `platypush.plugins.weather.openweathermap.WeatherOpenWeatherMapPlugin` plugin configured

`__init__`(*poll_seconds: int = 60*, ***kwargs*)

Parameters **`poll_seconds`** – How often the backend should check for updates (default: every minute).

1.70 `platypush.backend.websocket`

class `platypush.backend.websocket.`**`WebsocketBackend`**(*port=8765*, *bind_address='0.0.0.0'*, *ssl_cafile=None*, *ssl_capath=None*, *ssl_cert=None*, *ssl_key=None*, *client_timeout=0*, ***kwargs*)

Backend to communicate messages over a websocket medium.

Requires:

- **websockets** (`pip install websockets`)

`__init__`(*port=8765*, *bind_address='0.0.0.0'*, *ssl_cafile=None*, *ssl_capath=None*, *ssl_cert=None*, *ssl_key=None*, *client_timeout=0*, ***kwargs*)

Parameters

- **`port`** (*`int`*) – Listen port for the websocket server (default: 8765)
- **`bind_address`** – Bind address for the websocket server (default: 0.0.0.0, listen for any IP connection)

- **ssl_cert** (*str*) – Path to the certificate file if you want to enable SSL (default: None)
- **ssl_key** (*str*) – Path to the key file if you want to enable SSL (default: None)
- **ssl_cafile** (*str*) – Path to the certificate authority file if required by the SSL configuration (default: None)
- **ssl_capath** (*str*) – Path to the certificate authority directory if required by the SSL configuration (default: None)
- **client_timeout** – Timeout without any messages being received before closing a client connection. A zero timeout keeps the websocket open until an error occurs (default: 0, no timeout)

notify_web_clients(*event*)
Notify all the connected web clients (over websocket) of a new event

on_stop()
Callback invoked when the process stops

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

send_message(*msg*, ***kwargs*)
Sends a platypush.message.Message to a node. To be implemented in the derived classes. By default, if the Redis backend is configured then it will try to deliver the message to other consumers through the configured Redis main queue.

Parameters

- **msg** – The message to send
- **queue_name** – Send the message on a specific queue (default: the queue_name configured on the Redis backend)

1.71 platypush.backend.wiimote

class platypush.backend.wiimote.**WiimoteBackend**(*bdaddr=None*, *inactivity_timeout=300*, **args*, ***kwargs*)
Backend to communicate with a Nintendo WiiMote controller

Triggers:

- `platypush.message.event.Wiimote.WiimoteEvent` when the state of the Wiimote (battery, buttons, acceleration etc.) changes

Requires:

- **python3-wiimote** (follow instructions at https://github.com/azzra/python3-wiimote)

__init__(*bdaddr=None*, *inactivity_timeout=300*, **args*, ***kwargs*)

Parameters

- **bdaddr** (*str*) – If set, connect to this specific Wiimote physical address (example: 00:11:22:33:44:55)
- **inactivity_timeout** (*float*) – Number of seconds elapsed from the last Wiimote action before disconnecting the device (default: 300 seconds)

run()
Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.72 platypush.backend.zigbee.mqtt

class platypush.backend.zigbee.mqtt.**ZigbeeMqttBackend**(*host: Optional[str] = None, port: Optional[int] = None, base_topic='zigbee2mqtt', tls_cafile: Optional[str] = None, tls_certfile: Optional[str] = None, tls_keyfile: Optional[str] = None, tls_version: Optional[str] = None, tls_ciphers: Optional[str] = None, username: Optional[str] = None, password: Optional[str] = None, client_id: Optional[str] = None, *args, **kwargs*)

Listen for events on a zigbee2mqtt service.

Triggers:

- *platypush.message.event.zigbee.mqtt.ZigbeeMqttOnlineEvent* when the service comes online.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttOfflineEvent* when the service goes offline.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDevicePropertySetEvent* when the properties of a connected device change.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDevicePairingEvent* when a device is pairing.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceConnectedEvent* when a device connects to the network.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceBannedEvent* when a device is banned from the network.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceRemovedEvent* when a device is removed from the network.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceRemovedFailedEvent* when a request to remove a device from the network fails.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceWhitelistedEvent* when a device is whitelisted on the network.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceRenamedEvent* when a device is renamed on the network.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceBindEvent* when a device bind event occurs.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttDeviceUnbindEvent* when a device unbind event occurs.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttGroupAddedEvent* when a group is added.

- *platypush.message.event.zigbee.mqtt.ZigbeeMqttGroupAddedFailedEvent* when a request to add a new group fails.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttGroupRemovedEvent* when a group is removed.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttGroupRemovedFailedEvent* when a request to remove a group fails.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttGroupRemoveAllEvent* when all the devices are removed from a group.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttGroupRemoveAllFailedEvent* when a request to remove all the devices from a group fails.
- *platypush.message.event.zigbee.mqtt.ZigbeeMqttErrorEvent* when an internal error occurs on the zigbee2mqtt service.

Requires:

- **paho-mqtt** (`pip install paho-mqtt`)
- The *platypush.plugins.zigbee.mqtt.ZigbeeMqttPlugin* plugin configured.

`__init__`(*host: Optional[str] = None*, *port: Optional[int] = None*, *base_topic='zigbee2mqtt'*, *tls_cafile: Optional[str] = None*, *tls_certfile: Optional[str] = None*, *tls_keyfile: Optional[str] = None*, *tls_version: Optional[str] = None*, *tls_ciphers: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, *client_id: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **`host`** – MQTT broker host (default: host configured on the `zigbee.mqtt` plugin).
- **`port`** – MQTT broker port (default: 1883).
- **`base_topic`** – Prefix of the topics published by zigbee2mqtt (default: '`zigbee2mqtt`').
- **`tls_cafile`** – If TLS/SSL is enabled on the MQTT server and the certificate requires a certificate authority to authenticate it, *ssl_cafile* will point to the provided ca.crt file (default: None)
- **`tls_certfile`** – If TLS/SSL is enabled on the MQTT server and a client certificate it required, specify it here (default: None)
- **`tls_keyfile`** – If TLS/SSL is enabled on the MQTT server and a client certificate key it required, specify it here (default: None) :type tls_keyfile: str
- **`tls_version`** – If TLS/SSL is enabled on the MQTT server and it requires a certain TLS version, specify it here (default: None)
- **`tls_ciphers`** – If TLS/SSL is enabled on the MQTT server and an explicit list of supported ciphers is required, specify it here (default: None)
- **`username`** – Specify it if the MQTT server requires authentication (default: None)
- **`password`** – Specify it if the MQTT server requires authentication (default: None)
- **`client_id`** – MQTT client ID (default: `<device_id>-zigbee-mqtt`, to prevent clashes with the *platypush.backend.mqtt.MqttBackend* `client_id`.

`run`()

Starts the backend thread. To be implemented in the derived classes if the loop method isn't defined.

1.73 platypush.backend.zwave

class platypush.backend.zwave.**ZwaveBackend**(*device: str, config_path: Optional[str] = None, user_path: Optional[str] = None, ready_timeout: float = 10.0, *args, **kwargs*)

Start and manage a Z-Wave network.

If you are using a USB adapter and want a consistent naming for the device paths, you can use udev.

```
# Get the vendorID and productID of your device through lsusb.
# Then add a udev rule for it to link it e.g. to /dev/zwave.

cat <<EOF > /etc/udev/rules.d/92-zwave.rules
SUBSYSTEM=="tty", ATTRS{idVendor}=="0658", ATTRS{idProduct}=="0200", SYMLINK+=
↪"zwave"
EOF

# Restart the udev service
systemctl restart systemd-udevd.service
```

Triggers:

- *platypush.message.event.zwave.ZwaveNetworkReadyEvent* when the network is up and running.
- *platypush.message.event.zwave.ZwaveNetworkStoppedEvent* when the network goes down.
- *platypush.message.event.zwave.ZwaveNetworkResetEvent* when the network is reset.
- *platypush.message.event.zwave.ZwaveNetworkErrorEvent* when an error occurs on the network.
- *platypush.message.event.zwave.ZwaveNodeQueryCompletedEvent* when all the nodes on the network have been queried.
- *platypush.message.event.zwave.ZwaveNodeEvent* when a node attribute changes.
- *platypush.message.event.zwave.ZwaveNodeAddedEvent* when a node is added to the network.
- *platypush.message.event.zwave.ZwaveNodeRemovedEvent* when a node is removed from the network.
- *platypush.message.event.zwave.ZwaveNodeRenamedEvent* when a node is renamed.
- *platypush.message.event.zwave.ZwaveNodeReadyEvent* when a node is ready.
- *platypush.message.event.zwave.ZwaveNodeGroupEvent* when a node is associated/de-associated to a group.
- *platypush.message.event.zwave.ZwaveNodeSceneEvent* when a scene is set on a node.
- *platypush.message.event.zwave.ZwaveNodePollingEnabledEvent* when the polling is successfully turned on a node.
- *platypush.message.event.zwave.ZwaveNodePollingDisabledEvent* when the polling is successfully turned off a node.
- *platypush.message.event.zwave.ZwaveButtonCreatedEvent* when a button is added to the network.

- *platypush.message.event.zwave.ZwaveButtonRemovedEvent* when a button is removed from the network.
- *platypush.message.event.zwave.ZwaveButtonOnEvent* when a button is pressed.
- *platypush.message.event.zwave.ZwaveButtonOffEvent* when a button is released.
- *platypush.message.event.zwave.ZwaveValueAddedEvent* when a value is added to a node on the network.
- *platypush.message.event.zwave.ZwaveValueChangedEvent* when the value of a node on the network changes.
- *platypush.message.event.zwave.ZwaveValueRefreshedEvent* when the value of a node on the network is refreshed.
- *platypush.message.event.zwave.ZwaveValueRemovedEvent* when the value of a node on the network is removed.
- *platypush.message.event.zwave.ZwaveCommandEvent* when a command is received on the network.
- *platypush.message.event.zwave.ZwaveCommandWaitingEvent* when a command is waiting for a message to complete.

Requires:

- **python-openzwave** (`pip install python-openzwave`)

`__init__`(*device: str*, *config_path: Optional[str] = None*, *user_path: Optional[str] = None*, *ready_timeout: float = 10.0*, **args*, ***kwargs*)

Parameters

- **`device`** – Path to the Z-Wave adapter (e.g. /dev/ttyUSB0 or /dev/ttyACM0).
- **`config_path`** – Z-Wave configuration path (default: `<OPENZWAVE_PATH>/ozw_config`).
- **`user_path`** – Z-Wave user path where runtime and configuration files will be stored (default: `<PLATYPUSH_WORKDIR>/zwave`).
- **`ready_timeout`** – Network ready timeout in seconds (default: 60).

CHAPTER 2

Plugins

2.1 `platypush.plugins.adafruit.io`

class `platypush.plugins.adafruit.io.`**`AdafruitIoPlugin`**(*username*, *key*, *throttle_seconds=None*, ***kwargs*)

This plugin allows you to interact with the Adafruit IO <https://io.adafruit.com>, a cloud-based message queue and storage. You can send values to feeds on your Adafruit IO account and read the values of those feeds as well through any device.

Requires:

- **adafruit-io** (`pip install adafruit-io`)
- Redis server running and Redis backend configured if you want to enable throttling

Some example usages:

```
# Send the temperature value for a connected sensor to the "temperature" feed
{
    "type": "request",
    "action": "adafruit.io.send",
    "args": {
        "feed": "temperature",
        "value": 25.0
    }
}

# Receive the most recent temperature value
{
    "type": "request",
    "action": "adafruit.io.receive",
    "args": {
        "feed": "temperature"
    }
}
```

`__init__`(*username*, *key*, *throttle_seconds=None*, ***kwargs*)

Parameters

- **`username`** (*`str`*) – Your Adafruit username
- **`key`** (*`str`*) – Your Adafruit IO key
- **`throttle_seconds`** (*`float`*) – If set, then instead of sending the values directly over `send` the plugin will first collect all the samples within the specified period and then dispatch them to Adafruit IO. You may want to set it if you have data sources providing a lot of data points and you don't want to hit the throttling limitations of Adafruit.

`delete`(*feed*, *data_id*)
Delete a data point from a feed

Parameters

- **`feed`** (*`str`*) – Feed name
- **`data_id`** (*`str`*) – Data point ID to remove

`receive`(*feed*, *limit=1*)
Receive data from the specified Adafruit IO feed

Parameters

- **`feed`** (*`str`*) – Feed name
- **`limit`** (*`int`*) – Maximum number of data points to be returned. If None, all the values in the feed will be returned. Default: 1 (return most recent value)

`receive_next`(*feed*)
Receive the next unprocessed data point from a feed

Parameters **`feed`** (*`str`*) – Feed name

`receive_previous`(*feed*)
Receive the last processed data point from a feed

Parameters **`feed`** (*`str`*) – Feed name

`send`(*feed*, *value*, *enqueue=True*)
Send a value to an Adafruit IO feed

Parameters

- **`feed`** (*`str`*) – Feed name
- **`value`** (*`Numeric or string`*) – Value to send
- **`enqueue`** (*`bool`*) – If throttle_seconds is set, this method by default will append values to the throttling queue to be periodically flushed instead of sending the message directly. In such case, pass enqueue=False to override the behaviour and send the message directly instead.

`send_location_data`(*feed*, *lat*, *lon*, *ele*, *value*)
Send location data to an Adafruit IO feed

Parameters

- **`feed`** (*`str`*) – Feed name
- **`lat`** (*`float`*) – Latitude
- **`lon`** (*`float`*) – Longitude

- **ele** (*float*) – Elevation
- **value** (*Numeric or string*) – Value to send

2.2 platypush.plugins.alarm

class platypush.plugins.alarm.**AlarmPlugin**(***kwargs*)

Alarm/timer plugin.

Requires:

- The `platypush.backend.alarm.AlarmBackend` backend configured and enabled.

add(*when: str, actions: Optional[list] = None, name: Optional[str] = None, audio_file: Optional[str] = None, audio_volume: Union[int, float, None] = None, enabled: bool = True*) → str

Add a new alarm. NOTE: alarms that aren't configured in the `platypush.backend.alarm.AlarmBackend` will only run in the current session. If you want an alarm to be permanently stored, you should configure it in the alarm backend configuration. You may want to add an alarm dynamically if it's a one-time alarm instead

Parameters

- **when** – When the alarm should be executed. It can be either a cron expression (for recurrent alarms), or a datetime string in ISO format (for one-shot alarms/timers), or an integer representing the number of seconds before the alarm goes on (e.g. 300 for 5 minutes).
- **actions** – List of actions to be executed.
- **name** – Alarm name.
- **audio_file** – Path of the audio file to be played.
- **audio_volume** – Volume of the audio.
- **enabled** – Whether the new alarm should be enabled (default: True).

Returns The alarm name.

disable(*name: str*)

Disable an alarm. This will prevent the alarm from executing until re-enabled or until the application is restarted.

Parameters **name** – Alarm name.

dismiss()

Dismiss the alarm that is currently running.

enable(*name: str*)

Enable an alarm.

Parameters **name** – Alarm name.

get_alarms() → List[Dict[str, Any]]

Get the list of configured alarms.

Returns List of the alarms, sorted by next scheduled run.

snooze(*interval: Optional[float] = 300.0*)

Snooze the alarm that is currently running for the specified number of seconds. The alarm will stop and resume again later.

Parameters **interval** – Snooze seconds before playing the alarm again (default: 300).

2.3 platypush.plugins.arduino

class platypush.plugins.arduino.**ArduinoPlugin**(*board: Optional[str] = None, board_type: Optional[str] = None, baud_rate: int = 57600, analog_pins: Optional[Dict[str, int]] = None, digital_pins: Optional[Dict[str, int]] = None, timeout: float = 20.0, conv_functions: Optional[Dict[Union[str, int], Union[str, Callable]]] = None, **kwargs*)

Interact with an Arduino connected to the host machine over USB using the Firmata protocol.

You have two options to communicate with an Arduino-compatible board over USB:

- Use this plugin if you want to use the general-purpose Firmata protocol - in this case most of your processing logic will be on the host side and you can read/write data to the Arduino transparently.
- Use the `platypush.plugins.serial.SerialPlugin` if instead you want to run more custom logic on the Arduino and communicate back with the host computer through JSON formatted messages.

Download and flash the Standard Firmata firmware to the Arduino in order to use this plugin.

Requires:

- **pyfirmata2** (`pip install pyfirmata2`)

__init__(*board: Optional[str] = None, board_type: Optional[str] = None, baud_rate: int = 57600, analog_pins: Optional[Dict[str, int]] = None, digital_pins: Optional[Dict[str, int]] = None, timeout: float = 20.0, conv_functions: Optional[Dict[Union[str, int], Union[str, Callable]]] = None, **kwargs*)

Parameters

- **board** – Default board name or path (e.g. `COM3` on Windows or `/dev/ttyUSB0` on Unix). If not set then the plugin will attempt an auto-discovery.
- **board_type** – Default board type. It can be 'mega', 'due' or 'nano'. Leave empty for auto-detection.
- **baud_rate** – Default serial baud rate (default: 57600)
- **analog_pins** – Optional analog PINs map name->pin_number.
- **digital_pins** – Optional digital PINs map name->pin_number.
- **timeout** – Board communication timeout in seconds.
- **conv_functions** – Optional mapping of conversion functions to apply to the analog values read from a certain PIN. The key can either be the PIN number or the name as specified in `analog_pins`, the value can be a function that takes an argument and transforms it or its lambda string representation. Note that `analog_read` returns by default float values in the range [0.0, 1.0]. Example:

```yaml
arduino:
    board: /dev/ttyUSB0
    analog_pins:
        temperature: 1  # Analog PIN 1

    conv_functions:
        temperature: 'lambda t: t * 500.0'
```

analog_read(*pin: Union[int, str], board: Optional[str] = None, board_type: Optional[str] = None, baud_rate: Optional[int] = None, conv_function: Union[str, Callable, None] = None, timeout: Optional[int] = None*) → float

Read an analog value from a PIN.

Parameters

- **pin** – PIN number or configured name.
- **board** – Board path or name (default: default configured `board`).
- **board_type** – Board type. It can be 'mega', 'due' or 'nano' (default: configured `board_type`).
- **baud_rate** – Baud rate (default: default configured `baud_rate`).
- **conv_function** – Optional conversion function override to apply to the output. It can be either a function object or its lambda string representation (e.g. `lambda x: x*x`). Keep in mind that `analog_read` returns by default float values in the range `[0.0, 1.0]`.
- **timeout** – Communication timeout in seconds (default: default configured `timeout`).

analog_write(*pin: Union[int, str], value: float, board: Optional[str] = None, board_type: Optional[str] = None, baud_rate: Optional[int] = None, timeout: Optional[int] = None*)

Write a value to an analog PIN.

Parameters

- **pin** – PIN number or configured name.
- **value** – Voltage to be sent, a real number normalized between 0 and 1.
- **board** – Board path or name (default: default configured `board`).
- **board_type** – Board type. It can be 'mega', 'due' or 'nano' (default: configured `board_type`).
- **baud_rate** – Baud rate (default: default configured `baud_rate`).
- **timeout** – Communication timeout in seconds (default: default configured `timeout`).

digital_read(*pin: Union[int, str], board: Optional[str] = None, board_type: Optional[str] = None, baud_rate: Optional[int] = None, timeout: Optional[int] = None*) → bool

Read a digital value from a PIN.

Parameters

- **pin** – PIN number or configured name.
- **board** – Board path or name (default: default configured `board`).
- **board_type** – Board type. It can be 'mega', 'due' or 'nano' (default: configured `board_type`).
- **baud_rate** – Baud rate (default: default configured `baud_rate`).
- **timeout** – Communication timeout in seconds (default: default configured `timeout`).

digital_write(*pin: Union[int, str], value: bool, board: Optional[str] = None, board_type: Optional[str] = None, baud_rate: Optional[int] = None, timeout: Optional[int] = None*)

Write a value to a digital PIN.

Parameters

- **pin** – PIN number or configured name.

- **value** – True (HIGH) or False (LOW).
- **board** – Board path or name (default: default configured `board`).
- **board_type** – Board type. It can be 'mega', 'due' or 'nano' (default: configured `board_type`).
- **baud_rate** – Baud rate (default: default configured `baud_rate`).
- **timeout** – Communication timeout in seconds (default: default configured `timeout`).

get_measurement(*board: Optional[str] = None*, *board_type: Optional[str] = None*, *baud_rate: Optional[int] = None*, *timeout: Optional[int] = None*) → Dict[str, float]

Get a measurement from all the configured PINs.

Parameters

- **board** – Board path or name (default: default configured `board`)
- **board_type** – Board type. It can be 'mega', 'due' or 'nano' (default: configured `board_type`).
- **baud_rate** – Baud rate (default: default configured `baud_rate`)
- **timeout** – Communication timeout in seconds (default: default configured `timeout`).

Returns dict, where the keys are either the configured names of the PINs (see `analog_pins` configuration) or all the analog PINs (names will be in the format 'A0..A7' in that case), and the values will be the real values measured, either normalized between 0 and 1 if no conversion functions were provided, or transformed through the configured `conv_functions`.

pwm_write(*pin: Union[int, str], value: float, board: Optional[str] = None, board_type: Optional[str] = None, baud_rate: Optional[int] = None, timeout: Optional[int] = None*)

Write a PWM value to a digital PIN.

Parameters

- **pin** – PIN number or configured name.
- **value** – PWM real value normalized between 0 and 1.
- **board** – Board path or name (default: default configured `board`).
- **board_type** – Board type. It can be 'mega', 'due' or 'nano' (default: configured `board_type`).
- **baud_rate** – Baud rate (default: default configured `baud_rate`).
- **timeout** – Communication timeout in seconds (default: default configured `timeout`).

class `platypush.plugins.arduino.`**BoardType**

An enumeration.

class `platypush.plugins.arduino.`**PinType**

An enumeration.

2.4 platypush.plugins.assistant

class `platypush.plugins.assistant.`**AssistantPlugin**(***kwargs*)

Base class for assistant plugins

is_detecting() → bool

Returns True if the asistant is detecting, False otherwise.

pause_detection()
: Put the assistant on pause. No new conversation events will be triggered.

resume_detection()
: Resume the assistant hotword detection from a paused state.

start_conversation(**args*, *language=None*, *tts_plugin=None*, *tts_args=None*, ***kwargs*)
: Start a conversation.

stop_conversation(**args*, ***kwargs*)
: Stop a conversation.

2.5 platypush.plugins.assistant.echo

class platypush.plugins.assistant.echo.**AssistantEchoPlugin**(*avs_config_file: str = None*, *audio_device: str = 'default'*, *audio_player: str = 'default'*, ***kwargs*)

Amazon Echo/Alexa assistant plugin.

In order to activate the Echo service on your device follow these steps:

1. Install avs (`pip install git+https://github.com/BlackLight/avs.git`)
2. Run the `alexa-auth` script. A local webservice will start on port 3000
3. If a browser instance doesn't open automatically then head to http://localhost:3000
4. Log in to your Amazon account
5. The required credentials will be stored to ~/.avs.json

Triggers:

- *platypush.message.event.assistant.ConversationStartEvent* when a new conversation starts
- *platypush.message.event.assistant.SpeechRecognizedEvent* when a new voice command is recognized
- *platypush.message.event.assistant.ConversationEndEvent* when a new conversation ends

Requires:

- **avs** (`pip install avs`)

__init__(*avs_config_file: str = None*, *audio_device: str = 'default'*, *audio_player: str = 'default'*, ***kwargs*)

Parameters

- **avs_config_file** – AVS credentials file - default: ~/.avs.json. If the file doesn't exist then an instance of the AVS authentication service will be spawned. You can login through an Amazon account either in the spawned browser window, if available, or by opening http://your-ip:3000 in the browser on another machine.
- **audio_device** – Name of the input audio device (default: 'default')
- **audio_player** – Player to be used for audio playback (default: 'default'). Supported values: 'mpv', 'mpg123', 'gstreamer'

start_conversation(***kwargs*)
Start a conversation.

stop_conversation()
Stop a conversation.

2.6 platypush.plugins.assistant.google

class platypush.plugins.assistant.google.**AssistantGooglePlugin**(***kwargs*)
Google assistant plugin. It acts like a wrapper around the *platypush.backend.assistant.google* backend to programmatically control the conversation status.

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

is_muted() → bool

Returns True if the microphone is muted, False otherwise.

send_text_query(*query: str*)
Send a text query to the assistant.

Parameters **query** – Query to be sent.

set_mic_mute(*muted: bool = True*)
Programmatically mute/unmute the microphone.

Parameters **muted** – Set to True or False.

start_conversation(***kwargs*)
Programmatically start a conversation with the assistant

stop_conversation()
Programmatically stop a running conversation with the assistant

toggle_mic_mute()
Toggle the mic mute state.

2.7 platypush.plugins.assistant.google.pushtotalk

class platypush.plugins.assistant.google.pushtotalk.**AssistantGooglePushtotalkPlugin**(*credentia*
oauthlib-
tool/crede
device_co
assistant/
language
US',
play_resp
tts_plugin
tts_args=
***kwargs*

Plugin for the Google Assistant push-to-talk API.

Triggers:

- *platypush.message.event.assistant.ConversationStartEvent* when a new conversation starts

- ***platypush.message.event.assistant.SpeechRecognizedEvent*** when a new voice command is recognized
- ***platypush.message.event.assistant.ConversationEndEvent*** when a new conversation ends

Requires:

- **tenacity** (`pip install tenacity`)
- **google-assistant-sdk** (`pip install google-assistant-sdk[samples]`)

`__init__`(*credentials_file='/home/docs/.config/google-oauthlib-tool/credentials.json', device_config='/home/docs/.config/googlesamples-assistant/device_config.json', language='en-US', play_response=True, tts_plugin=None, tts_args=None, **kwargs*)

Parameters

- **`credentials_file`** (*str*) – Path to the Google OAuth credentials file (default: ~/.config/google-oauthlib-tool/credentials.json). See https://developers.google.com/assistant/sdk/guides/library/python/embed/install-sample#generate_credentials for instructions to get your own credentials file.
- **`device_config`** (*str*) – Path to device_config.json. Register your device (see https://developers.google.com/assistant/sdk/guides/library/python/embed/register-device) and create a project, then run the pushtotalk.py script from googlesamples to create your device_config.json
- **`language`** (*str*) – Assistant language (default: en-US)
- **`play_response`** (*bool*) – If True (default) then the plugin will play the assistant response upon processed response. Otherwise nothing will be played - but you may want to handle the `ResponseEvent` manually.
- **`tts_plugin`** (*str*) – Optional text-to-speech plugin to be used to process response text.
- **`tts_args`** (*dict*) – Optional arguments for the TTS plugin `say` method.

`on_conversation_end`()
Conversation end handler

`on_conversation_start`()
Conversation start handler

`on_response`()
Response handler

`on_speech_recognized`()
Speech recognized handler

`on_volume_changed`()
Volume changed event

`set_mic_mute`(*muted: bool = True*)
Programmatically mute/unmute the microphone.

Parameters **`muted`** – Set to True or False.

`start_conversation`(**args, language: Optional[str] = None, tts_plugin: Optional[str] = None, tts_args: Optional[Dict[str, Any]] = None, **kwargs*)
Start a conversation

Parameters

- **`language`** – Language code override (default: default configured language).

- **tts_plugin** – Optional text-to-speech plugin to be used for rendering text.
- **tts_args** – Optional arguments for the TTS plugin say method.

Returns

A list of the interactions that happen within the conversation.

```
[
    {
        "request": "request 1",
        "response": "response 1"
    },
    {
        "request": "request 2",
        "response": "response 2"
    }
]
```

stop_conversation()

Stop a conversation.

2.8 platypush.plugins.autoremote

class platypush.plugins.autoremote.**AutoremotePlugin**(*devices=None, key=None, password=None, *args, **kwargs*)

This plugin allows you to send messages and notifications to an Android device that runs AutoRemote (https://joaoapps.com/autoremote/). You can also build custom actions to run on your Android device upon AutoRemote events using Tasker (https://tasker.joaoapps.com/).

Requires:

- **requests** (`pip install requests`)

__init__(*devices=None, key=None, password=None, *args, **kwargs*)

Parameters

- **devices** (*dict*) – Set this attribute if you want to control multiple AutoRemote devices. This will be a map in the format:

```
{
    'device_name': {
        'key': 'AUTOREMOTE_KEY',
        'password': 'DEVICE_PASSWORD'
    },
    ...
}
```

- **key** (*str*) – The key associated to your device. Open the link in your AutoRemote app and copy the key in the target URL. Set this value if you want to communicate with only one AutoRemote device.
- **password** (*str*) – AutoRemote password configured on the device (default: None). Set this value if you want to communicate with only one AutoRemote device.

send_message(*msg, key=None, password=None, devices=None, target=None, sender=None, ttl=None, group=None, *args, **kwargs*)

Sends a message to AutoRemote.

Parameters

- **msg** – Message to send
- **key** (*str*) – Set it if you want to override the default devices (default: None, message sent to all the configured devices)
- **password** (*str*) – Set it if you want to override the default password (default: None)
- **devices** (*list*) – Set it if you want to send the message to a specific list of configured devices (default: None, message sent to all the configured devices)
- **target** (*str*) – Message target (default: None)
- **sender** – Message sender (default: None)
- **ttl** (*int*) – Message time-to-live in seconds (default: None)
- **group** (*str*) – Message group name (default: None)

send_notification(*text=None, key=None, password=None, devices=None, title=None, target=None, sender=None, ttl=None, group=None, sound=None, vibration=None, url=None, id=None, action=None, icon=None, led=None, ledon=None, ledoff=None, picture=None, share=False, msg=None, action1=None, action1_name=None, action1_icon=None, action2=None, action2_name=None, action2_icon=None, action3=None, action3_name=None, action3_icon=None, persistent=False, statusbar_icon=None, ticker=None, dismiss_on_touch=False, priority=0, number=None, content_info=None, subtext=None, max_progress=None, progress=None, indeterminate_progress=False, action_on_dismiss=None, cancel=False, *args, **kwargs*)

Sends a notification to AutoRemote. Click on your AutoRemote URL -> Send Notification for a detailed explanation of the attributes.

2.9 platypush.plugins.bluetooth

class platypush.plugins.bluetooth.**BluetoothPlugin**(*device_id: int = -1, **kwargs*)

Bluetooth plugin

Requires:

- **pybluez** (`pip install pybluez`)
- **pyobex** (`pip install git+https://github.com/BlackLight/PyOBEX`)

__init__(*device_id: int = -1, **kwargs*)

Parameters **device_id** – Default adapter device_id to be used (default: -1, auto)

close(*device: str = None, port: int = None, service_uuid: str = None, service_name: str = None*)

Close an active bluetooth connection

Parameters

- **device** – Device address or name
- **port** – Port number
- **service_uuid** – Service UUID
- **service_name** – Service name

connect(*protocol=None*, *device: str = None*, *port: int = None*, *service_uuid: str = None*, *service_name: str = None*)

Connect to a bluetooth device. You can query the advertised services through `find_service`.

Parameters

- **protocol** – Supported values: either 'RFCOMM'/'L2CAP' (str) or bluetooth.RFCOMM/bluetooth.L2CAP int constants (int)
- **device** – Device address or name
- **port** – Port number
- **service_uuid** – Service UUID
- **service_name** – Service name

find_service(*name: str = None*, *addr: str = None*, *uuid: str = None*) → platypush.message.response.bluetooth.BluetoothLookupServiceResponse

Look up for a service published by a nearby bluetooth device. If all the parameters are null then all the published services on the nearby devices will be returned. See *:class:platypush.message.response.bluetoothBluetoothLookupServiceResponse* for response structure reference.

Parameters

- **name** – Service name
- **addr** – Service/device address
- **uuid** – Service UUID

get_measurement(*device_id: Optional[int] = None*, *duration: Optional[int] = 10*, **args*, ***kwargs*) → Dict[str, dict]

Wrapper for `scan` that returns bluetooth devices in a format usable by sensor backends.

Parameters

- **device_id** – Bluetooth adapter ID to use (default configured if None)
- **duration** – Scan duration in seconds

Returns Device address -> info map.

lookup_address(*name: str*, *timeout: int = 10*) → platypush.message.response.bluetooth.BluetoothLookupNameResponse

Look up the address of a nearby bluetooth device given the name

Parameters

- **name** – Device name
- **timeout** – Lookup timeout (default: 10 seconds)

lookup_name(*addr: str*, *timeout: int = 10*) → platypush.message.response.bluetooth.BluetoothLookupNameResponse

Look up the name of a nearby bluetooth device given the address

Parameters

- **addr** – Device address
- **timeout** – Lookup timeout (default: 10 seconds)

recv(*device: str*, *port: int*, *service_uuid: str = None*, *service_name: str = None*, *size: int = 1024*, *binary: bool = False*) → platypush.message.response.bluetooth.BluetoothResponse

Send data to an active bluetooth connection

Parameters

- **device** – Device address or name
- **port** – Port number
- **service_uuid** – Service UUID
- **service_name** – Service name
- **size** – Maximum number of bytes to be read
- **binary** – Set to true to return a base64-encoded binary string

scan(*device_id: Optional[int] = None*, *duration: int = 10*) → platypush.message.response.bluetooth.BluetoothScanResponse

Scan for nearby bluetooth devices

Parameters

- **device_id** – Bluetooth adapter ID to use (default configured if None)
- **duration** – Scan duration in seconds

send(*data*, *device: str = None*, *port: int = None*, *service_uuid: str = None*, *service_name: str = None*, *binary: bool = False*)

Send data to an active bluetooth connection

Parameters

- **data** – Data to be sent
- **device** – Device address or name
- **service_uuid** – Service UUID
- **service_name** – Service name
- **port** – Port number
- **binary** – Set to true if msg is a base64-encoded binary string

send_file(*filename: str*, *device: str*, *port: int = None*, *data=None*, *service_name='OBEX Object Push'*, *binary: bool = False*)

Send a local file to a device that exposes an OBEX Object Push service

Parameters

- **filename** – Path of the file to be sent
- **data** – Alternatively to a file on disk you can send raw (string or binary) content
- **device** – Device address or name
- **port** – Port number
- **service_name** – Service name
- **binary** – Set to true if data is a base64-encoded binary string

set_l2cap_mtu(*mtu: int*, *device: str = None*, *port: int = None*, *service_name: str = None*, *service_uuid: str = None*)

Set the L2CAP MTU (Maximum Transmission Unit) value for a connected bluetooth device. Both the devices usually use the same MTU value over a connection.

Parameters

- **device** – Device address or name
- **port** – Port number
- **service_uuid** – Service UUID

- **`service_name`** – Service name
- **`mtu`** – New MTU value

2.10 `platypush.plugins.bluetooth.ble`

class `platypush.plugins.bluetooth.ble.`**`BluetoothBlePlugin`**(*interface: str = 'hci0'*, ***kwargs*)

Bluetooth BLE (low-energy) plugin

Requires:

- **pybluez** (`pip install pybluez`)
- **gattlib** (`pip install gattlib`)

Note that the support for bluetooth low-energy devices on Linux requires:

- A bluetooth adapter compatible with the bluetooth 5.0 specification or higher;
- To run platypush with root privileges (which is usually a very bad idea), or to set the raw net capabilities on the Python executable (which is also a bad idea, because any Python script will be able to access the kernel raw network API, but it's probably better than running a network server that can execute system commands as root user). If you don't want to set special permissions on the main Python executable and you want to run the bluetooth LTE plugin then the advised approach is to install platypush in a virtual environment and set the capabilities on the venv python executable, or run your platypush instance in Docker.

  You can set the capabilities on the Python executable through the following shell command:

  ```
  [sudo] setcap 'cap_net_raw,cap_net_admin+eip' /path/to/your/python
  ```

`__init__`(*interface: str = 'hci0'*, ***kwargs*)

Parameters **`interface`** – Default adapter device to be used (default: 'hci0')

`connect`(*device: str*, *interface: str = None*, *wait: bool = True*, *channel_type: str = 'public'*, *security_level: str = 'low'*, *psm: int = 0*, *mtu: int = 0*, *timeout: float = 10.0*)

Connect to a bluetooth LE device

Parameters

- **`device`** – Device address to connect to
- **`interface`** – Bluetooth adapter name to use (default configured if None)
- **`wait`** – If True then wait for the connection to be established before returning (no timeout)
- **`channel_type`** – Channel type, usually 'public' or 'random'
- **`security_level`** – Security level - possible values: ['low', 'medium', 'high']
- **`psm`** – PSM value (default: 0)
- **`mtu`** – MTU value (default: 0)
- **`timeout`** – Connection timeout if wait is not set (default: 10 seconds)

`disconnect`(*device: str*)

Disconnect from a connected device

Parameters **`device`** – Device address

discover_characteristics(*device: str*, *interface: str = None*, ***kwargs*) → platypush.message.response.bluetooth.BluetoothDiscoverCharacteristicsResponse

Discover the characteristics of a LE bluetooth device

Parameters

- **device** – Device address to connect to
- **interface** – Bluetooth adapter name to use (default configured if None)
- **kwargs** – Extra arguments to be passed to `connect()`

discover_primary(*device: str*, *interface: str = None*, ***kwargs*) → platypush.message.response.bluetooth.BluetoothDiscoverPrimaryResponse

Discover the primary services advertised by a LE bluetooth device

Parameters

- **device** – Device address to connect to
- **interface** – Bluetooth adapter name to use (default configured if None)
- **kwargs** – Extra arguments to be passed to `connect()`

get_measurement(*interface: Optional[str] = None*, *duration: Optional[int] = 10*, **args*, ***kwargs*) → Dict[str, dict]

Wrapper for `scan` that returns bluetooth devices in a format usable by sensor backends.

Parameters

- **interface** – Bluetooth adapter name to use (default configured if None)
- **duration** – Scan duration in seconds

Returns Device address -> info map.

read(*device: str*, *interface: str = None*, *uuid: str = None*, *handle: int = None*, *binary: bool = False*, *disconnect_on_recv: bool = True*, ***kwargs*) → str

Read a message from a device

Parameters

- **device** – Device address to connect to
- **interface** – Bluetooth adapter name to use (default configured if None)
- **uuid** – Service UUID. Either the UUID or the device handle must be specified
- **handle** – Device handle. Either the UUID or the device handle must be specified
- **binary** – Set to true to return data as a base64-encoded binary string
- **disconnect_on_recv** – If True (default) disconnect when the response is received
- **kwargs** – Extra arguments to be passed to `connect()`

scan(*interface: Optional[str] = None*, *duration: int = 10*) → platypush.message.response.bluetooth.BluetoothScanResponse

Scan for nearby bluetooth low-energy devices

Parameters

- **interface** – Bluetooth adapter name to use (default configured if None)
- **duration** – Scan duration in seconds

write(*device: str*, *data*, *handle: int = None*, *interface: str = None*, *binary: bool = False*, *disconnect_on_recv: bool = True*, ***kwargs*) → str

Writes data to a device

Parameters

- **`device`** – Device address to connect to
- **`data`** – Data to be written (str or bytes)
- **`interface`** – Bluetooth adapter name to use (default configured if None)
- **`handle`** – Device handle. Either the UUID or the device handle must be specified
- **`binary`** – Set to true if data is a base64-encoded binary string
- **`disconnect_on_recv`** – If True (default) disconnect when the response is received
- **`kwargs`** – Extra arguments to be passed to `connect()`

2.11 `platypush.plugins.calendar`

class `platypush.plugins.calendar.`**`CalendarPlugin`**(*calendars=None*, **args*, ***kwargs*)

The CalendarPlugin allows you to keep track of multiple calendars (Google or iCal URLs) and get joined events from all of them.

Requires:

- **dateutil** (`pip install python-dateutil`)

`__init__`(*calendars=None*, **args*, ***kwargs*)

Parameters **`calendars`** (`list`) – List of calendars to be queried. Supported types so far: Google Calendar and iCal URLs.

Example format:

```
calendars = [
    {
        "type": "platypush.plugins.google.calendar.GoogleCalendarPlugin"
    },

    {
        "type": "platypush.plugins.calendar.ical.IcalCalendarPlugin",
        "url": "https://www.facebook.com/ical/u.php?uid=USER_ID&key=FB_KEY"
    },

    ...
]
```

`get_upcoming_events`(*max_results=10*)

Get a list of upcoming events merging all the available calendars.

Parameters **`max_results`** (`int`) – Maximum number of results to be returned (default: 10)

Returns platypush.message.Response – Response object with the list of events in the Google calendar API format.

Example:

```
output = [
    {
        "id": "123456abcdef",
        "kind": "calendar#event",
        "status": "confirmed",
```

(continues on next page)

(continued from previous page)

```
        "htmlLink": "...",
        "created": "2018-06-01T01:23:45.000Z",
        "updated": "2018-06-01T01:23:45.000Z",
        "creator": {
            "email": "...",
            "displayName": "...",
            "self": true
        },
        "organizer" {
            "email": "...",
            "displayName": "...",
            "self": true
        },
        "start": {
            "dateTime": "2018-06-02T10:00:00.000Z",
        },
        "end": {
            "dateTime": "2018-06-02T12:00:00.000Z",
        },
    },
    ...
]
```

2.12 `platypush.plugins.calendar.ical`

class `platypush.plugins.calendar.ical.`**`CalendarIcalPlugin`**(*url*, **args*, ***kwargs*)

iCal calendars plugin. Interact with remote calendars in iCal format.

Requires:

- **icalendar** (`pip install icalendar`)
- **python-dateutil** (`pip install python-dateutil`)

`__init__`(*url*, **args*, ***kwargs*)

Parameters **`url`** (`str`) – iCal URL to parse

`get_upcoming_events`(*max_results=10*, *only_participating=True*)

Get the upcoming events. See `get_upcoming_events()`.

2.13 `platypush.plugins.camera`

class `platypush.plugins.camera.`**Camera**(*info: platypush.plugins.camera.model.camera.CameraInfo, start_event: threading.Event = <threading.Event object at 0x7fc392c57c10>, stream_event: threading.Event = <threading.Event object at 0x7fc392bf2910>, capture_thread: Union[threading.Thread, NoneType] = None, stream_thread: Union[threading.Thread, NoneType] = None, stream: Union[platypush.plugins.camera.model.writer.StreamWriter, NoneType] = None, preview: Union[platypush.plugins.camera.model.writer.preview.PreviewWriter, NoneType] = None, file_writer: Union[platypush.plugins.camera.model.writer.FileVideoWriter, NoneType] = None*)

__init__(*info: platypush.plugins.camera.model.camera.CameraInfo, start_event: threading.Event = <threading.Event object>, stream_event: threading.Event = <threading.Event object>, capture_thread: Optional[threading.Thread] = None, stream_thread: Optional[threading.Thread] = None, stream: Optional[platypush.plugins.camera.model.writer.StreamWriter] = None, preview: Optional[platypush.plugins.camera.model.writer.preview.PreviewWriter] = None, file_writer: Optional[platypush.plugins.camera.model.writer.FileVideoWriter] = None*) → None

class `platypush.plugins.camera.`**CameraInfo**(*device: Union[str, int, NoneType], bind_address: Union[str, NoneType] = None, capture_timeout: float = 20.0, color_transform: Union[str, NoneType] = None, ffmpeg_bin: Union[str, NoneType] = None, fps: Union[float, NoneType] = None, frames_dir: Union[str, NoneType] = None, grayscale: Union[bool, NoneType] = None, horizontal_flip: bool = False, input_codec: Union[str, NoneType] = None, input_format: Union[str, NoneType] = None, listen_port: Union[int, NoneType] = None, output_codec: Union[str, NoneType] = None, output_format: Union[str, NoneType] = None, resolution: Union[Tuple[int, int], NoneType] = None, rotate: Union[float, NoneType] = None, scale_x: Union[float, NoneType] = None, scale_y: Union[float, NoneType] = None, stream_format: Union[str, NoneType] = None, vertical_flip: bool = False, warmup_frames: int = 0, warmup_seconds: float = 0.0*)

__init__(*device: Union[str, int, None], bind_address: Optional[str] = None, capture_timeout: float = 20.0, color_transform: Optional[str] = None, ffmpeg_bin: Optional[str] = None, fps: Optional[float] = None, frames_dir: Optional[str] = None, grayscale: Optional[bool] = None, horizontal_flip: bool = False, input_codec: Optional[str] = None, input_format: Optional[str] = None, listen_port: Optional[int] = None, output_codec: Optional[str] = None, output_format: Optional[str] = None, resolution: Optional[Tuple[int, int]] = None, rotate: Optional[float] = None, scale_x: Optional[float] = None, scale_y: Optional[float] = None, stream_format: Optional[str] = None, vertical_flip: bool = False, warmup_frames: int = 0, warmup_seconds: float = 0.0*) → None

exception platypush.plugins.camera.**CameraException**

class platypush.plugins.camera.**CameraPlugin**(*device: Union[str, int, None] = None, resolution: Tuple[int, int] = (640, 480), frames_dir: Optional[str] = None, warmup_frames: int = 5, warmup_seconds: Optional[float] = 0.0, capture_timeout: Optional[float] = 20.0, scale_x: Optional[float] = None, scale_y: Optional[float] = None, rotate: Optional[float] = None, grayscale: Optional[bool] = None, color_transform: Union[str, int, None] = None, fps: float = 16, horizontal_flip: bool = False, vertical_flip: bool = False, input_format: Optional[str] = None, output_format: Optional[str] = None, stream_format: str = 'mjpeg', listen_port: Optional[int] = 5000, bind_address: str = '0.0.0.0', ffmpeg_bin: str = 'ffmpeg', input_codec: Optional[str] = None, output_codec: Optional[str] = None, **kwargs*)

Abstract plugin to control camera devices.

If the *platypush.backend.http.HttpBackend* is enabled then the plugins that implement this class can expose two endpoints:

- **`http://host:8008/camera/<plugin>/photo<.extension>` to capture a photo from the camera, where** `.extension` can be `.jpg`, `.png` or `.bmp`.
- **`http://host:8008/camera/<plugin>/video<.extension>` to get a live feed from the camera, where** `.extension` can be `.mjpeg`, `.mkv`/`.webm`, `.mp4`/`.h264` or `.h265`.

Both the endpoints support the same parameters of the constructor of this class (e.g. `device`, `warmup_frames`, `duration` etc.) as `GET` parameters.

Requires:

- **Pillow** (`pip install Pillow`) [optional] default handler for image transformations.
- **wxPython (`pip install wxPython`) [optional] default handler for camera previews (`ffplay` will be** used as a fallback if `wxPython` is not installed).
- **ffmpeg** (see installation instructions for your OS) for rendering/streaming videos.

Triggers:

- ***platypush.message.event.camera.CameraRecordingStartedEvent*** when a new video recording/photo burst starts
- ***platypush.message.event.camera.CameraRecordingStoppedEvent*** when a video recording/photo burst ends
- ***platypush.message.event.camera.CameraVideoRenderedEvent*** when a sequence of captured is successfully rendered into a video
- ***platypush.message.event.camera.CameraPictureTakenEvent*** when a snapshot is captured and stored to an image file

`__init__`(*device: Union[str, int, None] = None, resolution: Tuple[int, int] = (640, 480), frames_dir: Optional[str] = None, warmup_frames: int = 5, warmup_seconds: Optional[float] = 0.0, capture_timeout: Optional[float] = 20.0, scale_x: Optional[float] = None, scale_y: Optional[float] = None, rotate: Optional[float] = None, grayscale: Optional[bool] = None, color_transform: Union[str, int, None] = None, fps: float = 16, horizontal_flip: bool = False, vertical_flip: bool = False, input_format: Optional[str] = None, output_format: Optional[str] = None, stream_format: str = 'mjpeg', listen_port: Optional[int] = 5000, bind_address: str = '0.0.0.0', ffmpeg_bin: str = 'ffmpeg', input_codec: Optional[str] = None, output_codec: Optional[str] = None, **kwargs*)

Parameters

- **`device`** – Identifier of the default capturing device.
- **`resolution`** – Default resolution, as a tuple of two integers.
- **`frames_dir`** – Directory where the camera frames will be stored (default: `~/.local/share/platypush/<plugin.name>/frames`)
- **`warmup_frames`** – Cameras usually take a while to adapt their luminosity and focus to the environment when taking a picture. This parameter allows you to specify the number of "warmup" frames to capture upon picture command before actually capturing a frame (default: 5 but you may want to calibrate this parameter for your camera)
- **`warmup_seconds`** – Number of seconds to wait before a picture is taken or the first frame of a video/sequence is captured (default: 0).
- **`capture_timeout`** – Maximum number of seconds to wait between the programmed termination of a capture session and the moment the device is released.
- **`scale_x`** – If set, the images will be scaled along the x axis by the specified factor
- **`scale_y`** – If set, the images will be scaled along the y axis by the specified factor
- **`color_transform`** – Color transformation to apply to the images.
- **`grayscale`** – Whether the output should be converted to grayscale.
- **`rotate`** – If set, the images will be rotated by the specified number of degrees
- **`fps`** – Frames per second (default: 25).
- **`horizontal_flip`** – If set, the images will be flipped on the horizontal axis.
- **`vertical_flip`** – If set, the images will be flipped on the vertical axis.
- **`listen_port`** – Default port to be used for streaming over TCP (default: 5000).
- **`bind_address`** – Default bind address for TCP streaming (default: 0.0.0.0, accept any connections).
- **`input_codec`** – Specify the ffmpeg video codec (`-vcodec`) used for the input.
- **`output_codec`** – Specify the ffmpeg video codec (`-vcodec`) to be used for encoding the output. For some ffmpeg output formats (e.g. `h264` and `rtp`) this may default to `libxvid`.
- **`input_format`** – Plugin-specific format/type for the input stream.
- **`output_format`** – Plugin-specific format/type for the output videos.
- **`ffmpeg_bin`** – Path to the ffmpeg binary (default: `ffmpeg`).
- **`stream_format`** – Default format for the output when streamed to a network device. Available:

  - `MJPEG` (default)
  - `H264` (over `ffmpeg`)
  - `H265` (over `ffmpeg`)
  - `MKV` (over `ffmpeg`)
  - `MP4` (over `ffmpeg`)

`capture_frame` (*device: platypush.plugins.camera.model.camera.Camera*, **args*, ***kwargs*)
Capture a frame from a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **`device`** – An initialized `platypush.plugins.camera.Camera` object.

`capture_image` (*image_file: str*, *preview: bool = False*, ***camera*) → str
Capture an image.

Parameters

- **`image_file`** – Path where the output image will be stored.
- **`camera`** – Camera parameters override - see constructors parameters.
- **`preview`** – Show a preview of the camera frames.

Returns The local path to the saved image.

`capture_preview` (*duration: Optional[float] = None*, *n_frames: Optional[int] = None*, ***camera*) → dict
Start a camera preview session.

Parameters

- **`duration`** – Preview duration (default: until `stop_capture()` is called).
- **`n_frames`** – Number of frames to display before closing (default: until `stop_capture()` is called).
- **`camera`** – Camera object properties.

Returns The status of the device.

`capture_sequence` (*duration: Optional[float] = None*, *n_frames: Optional[int] = None*, *preview: bool = False*, ***camera*) → str
Capture a sequence of frames from a camera and store them to a directory.

Parameters

- **`duration`** – Duration of the sequence in seconds (default: until `stop_capture()` is called).
- **`n_frames`** – Number of images to be captured (default: until `stop_capture()` is called).
- **`camera`** – Camera parameters override - see constructors parameters. `frames_dir` and `fps` in particular can be specifically tuned for `capture_sequence`.
- **`preview`** – Show a preview of the camera frames.

Returns The directory where the image files have been stored.

`capture_video` (*duration: Optional[float] = None*, *video_file: Optional[str] = None*, *preview: bool = False*, ***camera*) → Union[str, dict]
Capture a video.

Parameters

- **`duration`** – Record duration in seconds (default: None, record until `stop_capture`).

- **video_file** – If set, the stream will be recorded to the specified video file (default: None).
- **camera** – Camera parameters override - see constructors parameters.
- **preview** – Show a preview of the camera frames.

Returns If duration is specified, the method will wait until the recording is done and return the local path to the recorded resource. Otherwise, it will return the status of the camera device after starting it.

capturing_thread(*camera: platypush.plugins.camera.model.camera.Camera*, *duration: Optional[float] = None*, *video_file: Optional[str] = None*, *image_file: Optional[str] = None*, *n_frames: Optional[int] = None*, *preview: bool = False*, ***kwargs*)
Camera capturing thread.

Parameters

- **camera** – An initialized `platypush.plugins.camera.Camera` object.
- **duration** – Capturing session duration in seconds (default: until `stop_capture()` is called).
- **video_file** – If set, the session will be recorded to this output video file (video capture mode).
- **image_file** – If set, the output of the session will be a single image file (photo mode).
- **n_frames** – Number of frames to be captured (default: until `stop_capture()` is called).
- **preview** – Start a preview window.
- **kwargs** – Extra arguments to be passed to `capture_frame()`.

close_device(*camera: platypush.plugins.camera.model.camera.Camera*, *wait_capture: bool = True*) → None
Close and release a device.

static encode_frame(*frame*, *encoding: str = 'jpeg'*) → bytes
Encode a frame to a target type. The default implementation assumes that frame is a `PIL.Image` object.

Parameters

- **frame** – Image frame (default: a `PIL.Image` object).
- **encoding** – Image encoding (e.g. `jpeg`).

static flip_frame(*frame*, *horizontal_flip: bool = False*, *vertical_flip: bool = False*)
Frame flip logic. Does nothing unless implemented by a derived plugin.

Parameters

- **frame** – Image frame (default: a `PIL.Image` object).
- **horizontal_flip** – Flip along the horizontal axis.
- **vertical_flip** – Flip along the vertical axis.

open(*device: Union[str, int, None] = None*, *stream: bool = None*, ***info*) → platypush.plugins.camera.model.camera.Camera
Initialize and open a device using a context manager pattern.

Parameters

- **device** – Capture device by name, path or ID.

- **stream** – If set, the frames will be streamed to `camera.stream`.
- **info** – Camera parameters override - see constructors parameters.

Returns The initialized *platypush.plugins.camera.Camera* object.

open_device (*device: Union[str, int, None] = None, stream: bool = False, **params*) → platypush.plugins.camera.model.camera.Camera

Initialize and open a device.

Returns The initialized camera device.

Raises `platypush.plugins.camera.CaptureSessionAlreadyRunningException`

prepare_device (*device: platypush.plugins.camera.model.camera.Camera*)

Prepare a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized *platypush.plugins.camera.Camera* object.

release_device (*device: platypush.plugins.camera.model.camera.Camera*)

Release a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized *platypush.plugins.camera.Camera* object.

static rotate_frame (*frame, rotation: Union[float, int, None] = None*)

Frame rotation logic. The default implementation assumes that frame is a `PIL.Image` object.

Parameters

- **frame** – Image frame (default: a `PIL.Image` object).
- **rotation** – Rotation angle in degrees.

static scale_frame (*frame, scale_x: Optional[float] = None, scale_y: Optional[float] = None*)

Frame scaling logic. The default implementation assumes that frame is a `PIL.Image` object.

Parameters

- **frame** – Image frame (default: a `PIL.Image` object).
- **scale_x** – X-scale factor.
- **scale_y** – Y-scale factor.

start_camera (*camera: platypush.plugins.camera.model.camera.Camera, preview: bool = False, *args, **kwargs*)

Start a camera capture session.

Parameters

- **camera** – An initialized *platypush.plugins.camera.Camera* object.
- **preview** – Show a preview of the camera frames.

start_streaming (*duration: Optional[float] = None, stream_format: str = 'mkv', **camera*) → dict

Expose the video stream of a camera over a TCP connection.

Parameters

- **duration** – Streaming thread duration (default: until *stop_streaming()* is called).
- **stream_format** – Format of the output stream - e.g. `h264`, `mjpeg`, `mkv` etc. (default: `mkv`).
- **camera** – Camera object properties - see constructor parameters.

Returns The status of the device.

status(*device: Union[str, int, None] = None*)
Returns the status of the specified camera or all the active cameras if `device` is `None`.

stop_capture(*device: Union[str, int, None] = None*)
Stop any capturing session on the specified device.

Parameters **device** – Name/path/ID of the device to stop (default: all the active devices).

stop_streaming(*device: Union[str, int, None] = None*)
Stop a camera over TCP session.

Parameters **device** – Name/path/ID of the device to stop (default: all the active devices).

static store_frame(*frame*, *filepath: str*, *format: Optional[str] = None*)
Capture a frame to the filesystem using the `PIL` library - it can be overridden by derived classes.

Parameters

- **frame** – Frame object (default: a byte-encoded object or a `PIL.Image` object).
- **filepath** – Destination file.
- **format** – Output format.

take_picture(*image_file: str*, *preview: bool = False*, ***camera*) → str
Alias for *capture_image()*.

Parameters

- **image_file** – Path where the output image will be stored.
- **camera** – Camera parameters override - see constructors parameters.
- **preview** – Show a preview of the camera frames.

Returns The local path to the saved image.

to_grayscale(*frame*)
Convert a frame to grayscale. The default implementation assumes that frame is a `PIL.Image` object.

Parameters **frame** – Image frame (default: a `PIL.Image` object).

static transform_frame(*frame*, *color_transform*)
Frame color space (e.g. `RGB24`, `YUV` etc.) transform logic. Does nothing unless implemented by a derived plugin.

wait_capture(*camera: platypush.plugins.camera.model.camera.Camera*) → None
Wait until a capture session terminates.

Parameters **camera** – Camera object. `camera.info.capture_timeout` is used as a capture thread termination timeout if set.

exception `platypush.plugins.camera.`**CaptureAlreadyRunningException**(*device*)

__init__(*device*)
Initialize self. See help(type(self)) for accurate signature.

class `platypush.plugins.camera.`**StreamWriter**(**args*, *sock: Optional[IO] = None*, ***kwargs*)
Abstract class for camera streaming operations.

__init__(**args*, *sock: Optional[IO] = None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

close()
Close the channel.

encode(*image: PIL.Image.Image*) → bytes
Encode an image before sending it to the channel.

Parameters **image** – PIL Image object.

Returns The bytes-encoded representation of the frame.

write(*image: PIL.Image.Image*)
Write an image to the channel.

Parameters **img** – PIL Image instance.

2.14 platypush.plugins.camera.android.ipcam

class platypush.plugins.camera.android.ipcam.**CameraAndroidIpcamPlugin**(*host: Optional[str] = None, port: Optional[int] = 8080, username: Optional[str] = None, password: Optional[str] = None, timeout: int = 10, ssl: bool = True, cameras: Optional[Dict[str, Dict[str, Any]]] = None, **kwargs*)

Plugin to control remote Android cameras over IPCam.

`__init__`(*host: Optional[str] = None*, *port: Optional[int] = 8080*, *username: Optional[str] = None*, *password: Optional[str] = None*, *timeout: int = 10*, *ssl: bool = True*, *cameras: Optional[Dict[str*, *Dict[str*, *Any]]] = None*, ***kwargs*)

Parameters

- **`host`** – Camera host name or address
- **`port`** – Camera port
- **`username`** – Camera username, if set
- **`password`** – Camera password, if set
- **`timeout`** – Connection timeout
- **`ssl`** – Use HTTPS instead of HTTP
- **`cameras`** – Alternatively, you can specify a list of IPCam cameras as a name->dict mapping. The keys will be unique names used to identify your cameras, the values will contain dictionaries containing *host, 'port*, *username*, *password*, *timeout* and *ssl* attributes for each camera.

`change_setting`(*key: str, value: Union[str, int, bool], camera: Union[int, str] = None*) → bool
Change a setting. :param key: Setting name :param value: Setting value :param camera: Camera index or configured name :return: True on success, False otherwise

`set_focus`(*activate: bool = True*, *camera: Union[int*, *str] = None*) → bool
Enable/disable the focus.

`set_front_facing_camera`(*activate: bool = True*, *camera: Union[int*, *str] = None*) → bool
Enable/disable the front-facing camera.

`set_gps`(*activate: bool = True*, *camera: Union[int*, *str] = None*) → bool
Enable/disable GPS.

`set_motion_detect`(*activate: bool = True*, *camera: Union[int*, *str] = None*) → bool
Enable/disable motion detect.

`set_night_vision`(*activate: bool = True*, *camera: Union[int*, *str] = None*) → bool
Enable/disable night vision.

`set_orientation`(*orientation: str = 'landscape'*, *camera: Union[int*, *str] = None*) → bool
Set video orientation.

`set_overlay`(*activate: bool = True*, *camera: Union[int*, *str] = None*) → bool
Enable/disable video overlay.

`set_quality`(*quality: int = 100*, *camera: Union[int*, *str] = None*) → bool
Set video quality.

`set_scenemode`(*scenemode: str = 'auto'*, *camera: Union[int*, *str] = None*) → bool
Set video orientation.

`set_torch`(*activate: bool = True*, *camera: Union[int*, *str] = None*) → bool
Enable/disable the torch.

`set_zoom`(*zoom: float*, *camera: Union[int*, *str] = None*) → bool
Set the zoom level.

`start_recording`(*tag: Optional[str] = None*, *camera: Union[int*, *str] = None*) → bool
Start recording.

`status`(*camera: Union[int*, *str] = None*) → platypush.message.response.camera.android.AndroidCameraStatusListResponse

Parameters **`camera`** – Camera index or name (default: status of all the cameras)

Returns True if the camera is available, False otherwise

stop_recording (*camera: Union[int, str] = None*) → bool
Stop recording.

take_picture (*image_file: str, camera: Union[int, str] = None*) → platypush.message.response.camera.android.AndroidCameraPictureResponse
Take a picture and save it on the local device.

2.15 `platypush.plugins.camera.cv`

class platypush.plugins.camera.cv.**CameraCvPlugin** (*color_transform: Optional[str] = 'COLOR_BGR2RGB', video_type: str = 'XVID', video_writer: str = 'ffmpeg', **kwargs*)

Plugin to control generic cameras over OpenCV.

Requires:

- **opencv** (`pip install opencv-python`)
- **Pillow** (`pip install Pillow`)

__init__ (*color_transform: Optional[str] = 'COLOR_BGR2RGB', video_type: str = 'XVID', video_writer: str = 'ffmpeg', **kwargs*)

Parameters

- **device** – Device ID (0 for the first camera, 1 for the second etc.) or path (e.g. `/dev/video0`).
- **video_type** – Default video type to use when exporting captured frames to camera (default: 0, infers the type from the video file extension). See here for a reference on the supported types (e.g. 'MJPEG', 'XVID', 'H264', 'X264', 'AVC1' etc.)
- **color_transform** – Color transformation to apply to the captured frames. See https://docs.opencv.org/3.2.0/d7/d1b/group__imgproc__misc.html for a full list of supported color transformations (default: "`COLOR_RGB2BGR`")
- **video_writer** – Class to be used to write frames to a video file. Supported values:
  - **`ffmpeg`: Use the FFmpeg writer (default, and usually more reliable - it requires `ffmpeg`** installed).
  - `cv`: Use the native OpenCV writer.

  The FFmpeg video writer requires `scikit-video` (`pip install scikit-video`) and `ffmpeg`.
- **kwargs** – Extra arguments to be passed up to `platypush.plugins.camera.CameraPlugin`.

capture_frame (*camera: platypush.plugins.camera.model.camera.Camera, *args, **kwargs*)
Capture a frame from a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters device – An initialized `platypush.plugins.camera.Camera` object.

prepare_device (*device: platypush.plugins.camera.model.camera.Camera*)
Prepare a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters device – An initialized `platypush.plugins.camera.Camera` object.

release_device(*device: platypush.plugins.camera.model.camera.Camera*)
Release a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

static transform_frame(*frame, color_transform: Union[str, int]*)
Frame color space (e.g. `RGB24`, `YUV` etc.) transform logic. Does nothing unless implemented by a derived plugin.

2.16 platypush.plugins.camera.ffmpeg

class platypush.plugins.camera.ffmpeg.**CameraFfmpegPlugin**(*device: Optional[str] = '/dev/video0', input_format: str = 'v4l2', ffmpeg_args: Tuple[str] = (), **opts*)

Plugin to interact with a camera over FFmpeg.

Requires:

- **ffmpeg** package installed on the system.

__init__(*device: Optional[str] = '/dev/video0', input_format: str = 'v4l2', ffmpeg_args: Tuple[str] = (), **opts*)

Parameters

- **device** – Path to the camera device (default: `/dev/video0`).
- **input_format** – FFmpeg input format for the the camera device (default: `v4l2`).
- **ffmpeg_args** – Extra options to be passed to the FFmpeg executable.
- **opts** – Camera options - see constructor of `platypush.plugins.camera.CameraPlugin`.

capture_frame(*camera: platypush.plugins.camera.ffmpeg.model.FFmpegCamera, *args, **kwargs*) → Optional[PIL.Image.Image]
Capture a frame from a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

prepare_device(*camera: platypush.plugins.camera.ffmpeg.model.FFmpegCamera*) → subprocess.Popen
Prepare a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

release_device(*camera: platypush.plugins.camera.ffmpeg.model.FFmpegCamera*)
Release a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

start_camera(*camera: platypush.plugins.camera.ffmpeg.model.FFmpegCamera, preview: bool = False, *args, **kwargs*)
Start a camera capture session.

Parameters

- **camera** – An initialized `platypush.plugins.camera.Camera` object.
- **preview** – Show a preview of the camera frames.

wait_capture(*camera: platypush.plugins.camera.ffmpeg.model.FFmpegCamera*) → None
Wait until a capture session terminates.

Parameters **camera** – Camera object. `camera.info.capture_timeout` is used as a capture thread termination timeout if set.

2.17 platypush.plugins.camera.gstreamer

class platypush.plugins.camera.gstreamer.**CameraGstreamerPlugin**(*device: Optional[str] = '/dev/video0'*, ***opts*)

Plugin to interact with a camera over GStreamer.

Requires:

- **gst-python** (`pip install gst-python`)

__init__(*device: Optional[str] = '/dev/video0'*, ***opts*)

Parameters

- **device** – Path to the camera device (default `/dev/video0`).
- **opts** – Camera options - see constructor of `platypush.plugins.camera.CameraPlugin`.

capture_frame(*camera: platypush.plugins.camera.gstreamer.model.GStreamerCamera*, **args*, ***kwargs*) → Optional[PIL.Image.Image]
Capture a frame from a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

prepare_device(*camera: platypush.plugins.camera.gstreamer.model.GStreamerCamera*) → platypush.common.gstreamer.Pipeline
Prepare a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

release_device(*camera: platypush.plugins.camera.gstreamer.model.GStreamerCamera*)
Release a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

start_camera(*camera: platypush.plugins.camera.gstreamer.model.GStreamerCamera*, *preview: bool = False*, **args*, ***kwargs*)
Start a camera capture session.

Parameters

- **camera** – An initialized `platypush.plugins.camera.Camera` object.
- **preview** – Show a preview of the camera frames.

2.18 `platypush.plugins.camera.ir.mlx90640`

class platypush.plugins.camera.ir.mlx90640.**CameraIrMlx90640Plugin**(*rawrgb_path: Optional[str] = None, resolution: Tuple[int, int] = (32, 24), warmup_frames: Optional[int] = 5, **kwargs*)

Plugin to interact with a ML90640 infrared thermal camera.

In order to use this plugin you'll need to download and compile the mlx90640 C++ bindings and examples for the device. Instructions on Raspbian:

```
# Install the dependencies
$ [sudo] apt-get install libi2c-dev
$ cd $PLATYPUSH_SRC_DIR
$ git submodule init
$ git submodule update
$ cd platypush/plugins/camera/ir/mlx90640/lib
$ make clean
$ make bcm2835
$ make examples/rawrgb I2C_MODE=LINUX
```

Requires:

- **mlx90640-library** installation (see instructions above)
- **PIL** image library (`pip install Pillow`)

__init__(*rawrgb_path: Optional[str] = None, resolution: Tuple[int, int] = (32, 24), warmup_frames: Optional[int] = 5, **kwargs*)

Parameters

- **rawrgb_path** – Specify it if the rawrgb executable compiled from https://github.com/pimoroni/mlx90640-library is in another folder than *<directory of this file>/lib/examples*.
- **resolution** – Device resolution (default: 32x24).
- **warmup_frames** – Number of frames to be skipped on sensor initialization/warmup (default: 2).
- **kwargs** – Extra parameters to be passed to `platypush.plugins.camera.CameraPlugin`.

capture(*output_file=None, *args, **kwargs*)
Back-compatibility alias for `capture_image()`.

capture_frame(*device: platypush.plugins.camera.model.camera.Camera, *args, **kwargs*)
Capture a frame from a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

prepare_device(*device: platypush.plugins.camera.model.camera.Camera*)
Prepare a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

release_device(*device: platypush.plugins.camera.model.camera.Camera*)
Release a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

to_grayscale(*image*)
Convert a frame to grayscale. The default implementation assumes that frame is a `PIL.Image` object.

Parameters **frame** – Image frame (default: a `PIL.Image` object).

2.19 platypush.plugins.camera.pi

class platypush.plugins.camera.pi.**CameraPiPlugin**(*device: int = 0, fps: float = 30.0, warmup_seconds: float = 2.0, sharpness: int = 0, contrast: int = 0, brightness: int = 50, video_stabilization: bool = False, iso: int = 0, exposure_compensation: int = 0, exposure_mode: str = 'auto', meter_mode: str = 'average', awb_mode: str = 'auto', image_effect: str = 'none', led_pin: Optional[int] = None, color_effects: Union[str, List[str], None] = None, zoom: Tuple[float, float, float, float] = (0.0, 0.0, 1.0, 1.0), **camera*)

Plugin to control a Pi camera.

Requires:

- **picamera** (`pip install picamera`)
- **numpy** (`pip install numpy`)
- **Pillow** (`pip install Pillow`)

__init__(*device: int = 0, fps: float = 30.0, warmup_seconds: float = 2.0, sharpness: int = 0, contrast: int = 0, brightness: int = 50, video_stabilization: bool = False, iso: int = 0, exposure_compensation: int = 0, exposure_mode: str = 'auto', meter_mode: str = 'average', awb_mode: str = 'auto', image_effect: str = 'none', led_pin: Optional[int] = None, color_effects: Union[str, List[str], None] = None, zoom: Tuple[float, float, float, float] = (0.0, 0.0, 1.0, 1.0), **camera*)
See https://www.raspberrypi.org/documentation/usage/camera/python/README.md for a detailed reference about the Pi camera options.

Parameters **camera** – Options for the base camera plugin (see `platypush.plugins.camera.CameraPlugin`).

capture_frame(*camera: platypush.plugins.camera.model.camera.Camera*, **args*, ***kwargs*)
Capture a frame from a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

capture_preview(*duration: Optional[float] = None*, *n_frames: Optional[int] = None*, ***camera*) → dict
Start a camera preview session.

Parameters

- **duration** – Preview duration (default: until `stop_capture()` is called).
- **n_frames** – Number of frames to display before closing (default: until `stop_capture()` is called).
- **camera** – Camera object properties.

Returns The status of the device.

prepare_device(*device: platypush.plugins.camera.pi.model.PiCamera*)
Prepare a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

release_device(*device: platypush.plugins.camera.pi.model.PiCamera*)
Release a device using the plugin-specific logic - to be implemented by the derived classes.

Parameters **device** – An initialized `platypush.plugins.camera.Camera` object.

start_preview(*camera: platypush.plugins.camera.model.camera.Camera*)
Start camera preview.

start_streaming(*duration: Optional[float] = None*, *stream_format: str = 'h264'*, ***camera*) → dict
Expose the video stream of a camera over a TCP connection.

Parameters

- **duration** – Streaming thread duration (default: until `stop_streaming()` is called).
- **stream_format** – Format of the output stream - e.g. `h264`, `mjpeg`, `mkv` etc. (default: `mkv`).
- **camera** – Camera object properties - see constructor parameters.

Returns The status of the device.

stop_preview(*camera: platypush.plugins.camera.model.camera.Camera*)
Stop camera preview.

2.20 platypush.plugins.chat.telegram

class platypush.plugins.chat.telegram.**ChatTelegramPlugin**(*api_token: str*, ***kwargs*)
Plugin to programmatically send Telegram messages through a Telegram bot. In order to send messages to contacts, groups or channels you'll first need to register a bot. To do so:

1. Open a Telegram conversation with the @BotFather.
2. Send `/start` followed by `/newbot`. Choose a display name and a username for your bot.
3. Copy the provided API token in the configuration of this plugin.
4. Open a conversation with your newly created bot.

Requires:

- **python-telegram-bot** (`pip install python-telegram-bot`)

`__init__`(*api_token: str*, ***kwargs*)

Parameters **`api_token`** – API token as returned by the @BotFather

`delete_chat_photo`(*chat_id: Union[str, int], timeout: int = 20*)
Delete the photo of a channel/group.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **`timeout`** – Request timeout (default: 20 seconds)

`get_chat`(*chat_id: Union[int, str], timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramChatResponse
Get the info about a Telegram chat.

Parameters

- **`chat_id`** – Chat ID.
- **`timeout`** – Upload timeout (default: 20 seconds).

`get_chat_administrators`(*chat_id: Union[int, str], timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramUsersResponse
Get the list of the administrators of a chat.

Parameters

- **`chat_id`** – Chat ID.
- **`timeout`** – Upload timeout (default: 20 seconds).

`get_chat_members_count`(*chat_id: Union[int, str], timeout: int = 20*) → int
Get the number of users in a chat.

Parameters

- **`chat_id`** – Chat ID.
- **`timeout`** – Upload timeout (default: 20 seconds).

`get_chat_user`(*chat_id: Union[int, str], user_id: int, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramUserResponse
Get the info about a user connected to a chat.

Parameters

- **`chat_id`** – Chat ID.
- **`user_id`** – User ID.
- **`timeout`** – Upload timeout (default: 20 seconds).

`get_file`(*file_id: str*, *timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramFileResponse
Get the info and URL of an uploaded file by file_id.

Parameters

- **`file_id`** – File ID.
- **`timeout`** – Upload timeout (default: 20 seconds).

kick_chat_member (*chat_id: Union[str, int], user_id: int, until_date: Optional[datetime.datetime] = None, timeout: int = 20*)

Kick a user from a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **user_id** – Unique user ID.
- **until_date** – End date for the ban.
- **timeout** – Request timeout (default: 20 seconds)

leave_chat (*chat_id: Union[str, int], timeout: int = 20*)

Leave a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **timeout** – Request timeout (default: 20 seconds)

pin_chat_message (*chat_id: Union[str, int], message_id: int, disable_notification: Optional[bool] = None, timeout: int = 20*)

Pin a message in a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **message_id** – Message ID.
- **disable_notification** – If True then no notification will be sent to the users.
- **timeout** – Request timeout (default: 20 seconds)

promote_chat_member (*chat_id: Union[str, int], user_id: int, can_change_info: Optional[bool] = None, can_post_messages: Optional[bool] = None, can_edit_messages: Optional[bool] = None, can_delete_messages: Optional[bool] = None, can_invite_users: Optional[bool] = None, can_restrict_members: Optional[bool] = None, can_promote_members: Optional[bool] = None, can_pin_messages: Optional[bool] = None, timeout: int = 20*)

Promote or demote a member.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".

- **user_id** – Unique user ID.
- **can_change_info** – Pass True if the user can change channel info.
- **can_post_messages** – Pass True if the user can post messages.
- **can_edit_messages** – Pass True if the user can edit messages.
- **can_delete_messages** – Pass True if the user can delete messages.
- **can_invite_users** – Pass True if the user can invite other users to the channel/group.
- **can_restrict_members** – Pass True if the user can restrict the permissions of other users.
- **can_promote_members** – Pass True if the user can promote mebmers.
- **can_pin_messages** – Pass True if the user can pin messages.
- **timeout** – Request timeout (default: 20 seconds)

send_animation(*chat_id: Union[str, int], file_id: Optional[int] = None, url: Optional[str] = None, path: Optional[str] = None, duration: Optional[int] = None, caption: Optional[str] = None, width: Optional[int] = None, height: Optional[int] = None, parse_mode: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send an animation (GIF or H.264/MPEG-4 AVC video without sound) to a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for “Telegram get channel/group chat_id”.
- **file_id** – Set it if the file already exists on Telegram servers and has a file_id. Note that you’ll have to specify either `file_id`, `url` or `path`.
- **url** – Set it if you want to send a file from a remote URL. Note that you’ll have to specify either `file_id`, `url` or `path`.
- **path** – Set it if you want to send a file from the local filesystem. Note that you’ll have to specify either `file_id`, `url` or `path`.
- **duration** – Duration in seconds.
- **caption** – Optional caption for the picture.
- **width** – Video width.
- **height** – Video height.
- **parse_mode** – Set to ‘Markdown’ or ‘HTML’ to send either Markdown or HTML content.
- **disable_notification** – If True then no notification will be sent to the users.
- **reply_to_message_id** – If set then the message will be sent as a response to the specified message.
- **timeout** – Upload timeout (default: 20 seconds)

send_audio(*chat_id: Union[str, int], file_id: Optional[int] = None, url: Optional[str] = None, path: Optional[str] = None, caption: Optional[str] = None, performer: Optional[str] = None, title: Optional[str] = None, duration: Optional[float] = None, parse_mode: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send audio to a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **file_id** – Set it if the file already exists on Telegram servers and has a file_id. Note that you'll have to specify either `file_id`, `url` or `path`.
- **url** – Set it if you want to send a file from a remote URL. Note that you'll have to specify either `file_id`, `url` or `path`.
- **path** – Set it if you want to send a file from the local filesystem. Note that you'll have to specify either `file_id`, `url` or `path`.
- **caption** – Optional caption for the picture.
- **performer** – Optional audio performer.
- **title** – Optional audio title.
- **duration** – Duration of the audio in seconds.
- **parse_mode** – Set to 'Markdown' or 'HTML' to send either Markdown or HTML content.
- **disable_notification** – If True then no notification will be sent to the users.
- **reply_to_message_id** – If set then the message will be sent as a response to the specified message.
- **timeout** – Upload timeout (default: 20 seconds)

send_contact(*chat_id: Union[str, int], phone_number: str, first_name: str, last_name: Optional[str] = None, vcard: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send a contact to a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **phone_number** – Phone number.
- **first_name** – First name.
- **last_name** – Last name.
- **vcard** – Additional contact info in vCard format (0-2048 bytes).

- **`disable_notification`** – If True then no notification will be sent to the users.
- **`reply_to_message_id`** – If set then the message will be sent as a response to the specified message.
- **`timeout`** – Upload timeout (default: 20 seconds)

`send_document`(*chat_id: Union[str, int], file_id: Optional[int] = None, url: Optional[str] = None, path: Optional[str] = None, filename: Optional[str] = None, caption: Optional[str] = None, parse_mode: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send a document to a chat.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **`file_id`** – Set it if the file already exists on Telegram servers and has a file_id. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`url`** – Set it if you want to send a file from a remote URL. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`path`** – Set it if you want to send a file from the local filesystem. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`filename`** – Name of the file as it will be shown in Telegram.
- **`caption`** – Optional caption for the picture.
- **`parse_mode`** – Set to 'Markdown' or 'HTML' to send either Markdown or HTML content.
- **`disable_notification`** – If True then no notification will be sent to the users.
- **`reply_to_message_id`** – If set then the message will be sent as a response to the specified message.
- **`timeout`** – Upload timeout (default: 20 seconds)

`send_location`(*chat_id: Union[str, int], latitude: float, longitude: float, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send a location to a chat.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **`latitude`** – Latitude
- **`longitude`** – Longitude
- **`disable_notification`** – If True then no notification will be sent to the users.

- **`reply_to_message_id`** – If set then the message will be sent as a response to the specified message.
- **`timeout`** – Upload timeout (default: 20 seconds)

`send_message`(*chat_id: Union[str, int], text: str, parse_mode: Optional[str] = None, disable_web_page_preview: bool = False, disable_notification: bool = False, reply_to_message_id: Optional[int] = None*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send a message to a chat.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **`text`** – Text to be sent.
- **`parse_mode`** – Set to 'Markdown' or 'HTML' to send either Markdown or HTML content.
- **`disable_web_page_preview`** – If True then web previews for URLs will be disabled.
- **`disable_notification`** – If True then no notification will be sent to the users.
- **`reply_to_message_id`** – If set then the message will be sent as a response to the specified message.

`send_photo`(*chat_id: Union[str, int], file_id: Optional[int] = None, url: Optional[str] = None, path: Optional[str] = None, caption: Optional[str] = None, parse_mode: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send a picture to a chat.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **`file_id`** – Set it if the file already exists on Telegram servers and has a file_id. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`url`** – Set it if you want to send a file from a remote URL. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`path`** – Set it if you want to send a file from the local filesystem. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`caption`** – Optional caption for the picture.
- **`parse_mode`** – Set to 'Markdown' or 'HTML' to send either Markdown or HTML content.
- **`disable_notification`** – If True then no notification will be sent to the users.

- **reply_to_message_id** – If set then the message will be sent as a response to the specified message.
- **timeout** – Upload timeout (default: 20 seconds)

send_venue(*chat_id: Union[str, int], latitude: float, longitude: float, title: str, address: str, foursquare_id: Optional[str] = None, foursquare_type: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send the address of a venue to a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **latitude** – Latitude
- **longitude** – Longitude
- **title** – Venue name.
- **address** – Venue address.
- **foursquare_id** – Foursquare ID.
- **foursquare_type** – Foursquare type.
- **disable_notification** – If True then no notification will be sent to the users.
- **reply_to_message_id** – If set then the message will be sent as a response to the specified message.
- **timeout** – Upload timeout (default: 20 seconds)

send_video(*chat_id: Union[str, int], file_id: Optional[int] = None, url: Optional[str] = None, path: Optional[str] = None, duration: Optional[int] = None, caption: Optional[str] = None, width: Optional[int] = None, height: Optional[int] = None, parse_mode: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send a video to a chat.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **file_id** – Set it if the file already exists on Telegram servers and has a file_id. Note that you'll have to specify either `file_id`, `url` or `path`.
- **url** – Set it if you want to send a file from a remote URL. Note that you'll have to specify either `file_id`, `url` or `path`.
- **path** – Set it if you want to send a file from the local filesystem. Note that you'll have to specify either `file_id`, `url` or `path`.
- **duration** – Duration in seconds.

- **caption** – Optional caption for the picture.
- **width** – Video width.
- **height** – Video height.
- **parse_mode** – Set to 'Markdown' or 'HTML' to send either Markdown or HTML content.
- **disable_notification** – If True then no notification will be sent to the users.
- **reply_to_message_id** – If set then the message will be sent as a response to the specified message.
- **timeout** – Upload timeout (default: 20 seconds)

send_video_note(*chat_id: Union[str, int], file_id: Optional[int] = None, url: Optional[str] = None, path: Optional[str] = None, duration: Optional[int] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send a video note to a chat. As of v.4.0, Telegram clients support rounded square mp4 videos of up to 1 minute long.

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **file_id** – Set it if the file already exists on Telegram servers and has a file_id. Note that you'll have to specify either `file_id`, `url` or `path`.
- **url** – Set it if you want to send a file from a remote URL. Note that you'll have to specify either `file_id`, `url` or `path`.
- **path** – Set it if you want to send a file from the local filesystem. Note that you'll have to specify either `file_id`, `url` or `path`.
- **duration** – Duration in seconds.
- **disable_notification** – If True then no notification will be sent to the users.
- **reply_to_message_id** – If set then the message will be sent as a response to the specified message.
- **timeout** – Upload timeout (default: 20 seconds)

send_voice(*chat_id: Union[str, int], file_id: Optional[int] = None, url: Optional[str] = None, path: Optional[str] = None, caption: Optional[str] = None, duration: Optional[float] = None, parse_mode: Optional[str] = None, disable_notification: bool = False, reply_to_message_id: Optional[int] = None, timeout: int = 20*) → platypush.message.response.chat.telegram.TelegramMessageResponse

Send audio to a chat as a voice file. For this to work, your audio must be in an .ogg file encoded with OPUS (other formats may be sent as Audio or Document).

Parameters

- **chat_id** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".

- **`file_id`** – Set it if the file already exists on Telegram servers and has a file_id. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`url`** – Set it if you want to send a file from a remote URL. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`path`** – Set it if you want to send a file from the local filesystem. Note that you'll have to specify either `file_id`, `url` or `path`.
- **`caption`** – Optional caption for the picture.
- **`duration`** – Duration of the voice in seconds.
- **`parse_mode`** – Set to 'Markdown' or 'HTML' to send either Markdown or HTML content.
- **`disable_notification`** – If True then no notification will be sent to the users.
- **`reply_to_message_id`** – If set then the message will be sent as a response to the specified message.
- **`timeout`** – Upload timeout (default: 20 seconds)

`set_chat_description`(*chat_id: Union[str, int], description: str, timeout: int = 20*)

Set the description of a channel/group.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **`description`** – New chat description.
- **`timeout`** – Request timeout (default: 20 seconds)

`set_chat_photo`(*chat_id: Union[str, int], path: str, timeout: int = 20*)

Set the photo of a channel/group.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for "Telegram get channel/group chat_id".
- **`path`** – Path of the new image.
- **`timeout`** – Request timeout (default: 20 seconds)

`set_chat_title`(*chat_id: Union[str, int], title: str, timeout: int = 20*)

Set the title of a channel/group.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for “Telegram get channel/group chat_id”.

- **`title`** – New chat title.
- **`timeout`** – Request timeout (default: 20 seconds)

`unban_chat_member`(*chat_id: Union[str, int], user_id: int, timeout: int = 20*)
Lift the ban from a chat member.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for “Telegram get channel/group chat_id”.
- **`user_id`** – Unique user ID.
- **`timeout`** – Request timeout (default: 20 seconds)

`unpin_chat_message`(*chat_id: Union[str, int], timeout: int = 20*)
Unpin the message of a chat.

Parameters

- **`chat_id`** – Chat ID. Can be either a numerical ID or a unique identifier in the format `@channelname`. In order to get your own Telegram chat_id open a conversation with @IDBot and type `/start` followed by `/getid`. Similar procedures also exist to get a group or channel chat_id - just Google for “Telegram get channel/group chat_id”.
- **`timeout`** – Request timeout (default: 20 seconds)

2.21 `platypush.plugins.clipboard`

class `platypush.plugins.clipboard.`**`ClipboardPlugin`**(***kwargs*)
Plugin to programmatically copy strings to your system clipboard and get the current clipboard content.

Requires:

- **pyperclip** (`pip install pyperclip`)

`copy`(*text*)
Copies a text to the OS clipboard

Parameters **`text`** (*`str`*) – Text to copy

`paste`()
Get the current content of the clipboard

2.22 `platypush.plugins.config`

class `platypush.plugins.config.`**`ConfigPlugin`**(***kwargs*)

2.23 `platypush.plugins.covid19`

class `platypush.plugins.covid19.`**`Covid19Plugin`**(*country: Union[str, List[str]] = 'world', **kwargs*)

Monitor the diffusion data of the COVID-19 pandemic by using the public API at https://api.covid19api.com.

`__init__`(*country: Union[str, List[str]] = 'world', **kwargs*)

Parameters **`country`** – Default country (or list of countries) to retrieve the stats for. It can either be the full country name or the country code. Special values:

- `world`: Get worldwide stats (default).
- `all`: Get all the available stats.

`data`(*country: Union[str, List[str], None] = None*) → List[Dict[str, Any]]

Get all the data for a country.

Parameters **`country`** – Default country override.

`summary`(*country: Union[str, List[str], None] = None*) → List[Dict[str, Any]]

Get the summary data for the world or a country.

Parameters **`country`** – Default country override.

2.24 `platypush.plugins.csv`

class `platypush.plugins.csv.`**`CsvPlugin`**(***kwargs*)

A plugin for managing CSV files.

`read`(*filename: str, delimiter: str = ', ', quotechar: Optional[str] = '"', start: int = 0, limit: Optional[int] = None, reverse: bool = False, has_header: bool = None, column_names: Optional[List[str]] = None, dialect: str = 'excel'*)

Gets the content of a CSV file.

Parameters

- **`filename`** – Path of the file.
- **`delimiter`** – Field delimiter (default: **`,`**).
- **`quotechar`** – Quote character (default: **`"`**).
- **`start`** – (Zero-based) index of the first line to be read (starting from the last if `reverse` is True) (default: 0).
- **`limit`** – Maximum number of lines to be read (default: all).
- **`reverse`** – If True then the lines will be read starting from the last (default: False).
- **`has_header`** – Set to True if the first row of the file is a header, False if the first row isn't expected to be a header (default: None, the method will scan the first chunk of the file and estimate whether the first line is a header).
- **`column_names`** – Specify if the file has no header or you want to override the column names.
- **`dialect`** – CSV dialect (default: `excel`).

static **`reversed_blocks`**(*f: TextIO, blocksize=4096*)

Generate blocks of file's contents in reverse order.

write(*filename: str, rows: List[Union[List[Any], Dict[str, Any]]], truncate: bool = False, delimiter: str = ',', quotechar: Optional[str] = '"', dialect: str = 'excel'*)
Writes lines to a CSV file.

Parameters

- **filename** – Path of the CSV file.
- **rows** – Rows to write. It can be a list of lists or a key->value dictionary where the keys match the names of the columns. If the rows are dictionaries then a header with the column names will be written to the file if not available already, otherwise no header will be written.
- **truncate** – If True then any previous file content will be removed, otherwise the new rows will be appended to the file (default: False).
- **delimiter** – Field delimiter (default: `,`).
- **quotechar** – Quote character (default: `"`).
- **dialect** – CSV dialect (default: `excel`).

2.25 `platypush.plugins.db`

class `platypush.plugins.db.`**DbPlugin**(*engine=None*, **args*, ***kwargs*)
Database plugin. It allows you to programmatically select, insert, update and delete records on a database backend through requests, procedures and event hooks.

Requires:

- **sqlalchemy** (`pip install sqlalchemy`)

__init__(*engine=None*, **args*, ***kwargs*)

Parameters

- **engine** (*str*) – Default SQLAlchemy connection engine string (e.g. `sqlite:///:memory:` or `mysql://user:pass@localhost/test`) that will be used. You can override the default engine in the db actions.
- **args** – Extra arguments that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)
- **kwargs** – Extra kwargs that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)

delete(*table*, *records*, *engine=None*, **args*, ***kwargs*)
Deletes records from a table.

Parameters

- **table** (*str*) – Table name
- **records** (*list*) – Records to be deleted, as a list of dictionaries
- **engine** (*str*) – Engine to be used (default: default class engine)
- **args** – Extra arguments that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)
- **kwargs** – Extra kwargs that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)

Example:

Request:

```
{
    "type": "request",
    "target": "your_host",
    "action": "db.delete",
    "args": {
        "table": "table",
        "engine": "sqlite:///:memory:",
        "records": [
            { "id": 1 },
            { "id": 2 }
        ]
    }
}
```

execute (*statement*, *engine=None*, **args*, ***kwargs*)

Executes a raw SQL statement.

> **Warning:** Avoid calling this method directly if possible. Use `insert`, `update` and `delete` methods instead if possible. Don't use this method if you need to select records, use the `select` method instead, as this method is mostly meant to execute raw SQL without returning anything.

Parameters

- **statement** (*str*) – SQL to be executed
- **engine** (*str*) – Engine to be used (default: default class engine)
- **args** – Extra arguments that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)
- **kwargs** – Extra kwargs that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)

insert (*table*, *records*, *engine=None*, *key_columns=None*, *on_duplicate_update=False*, **args*, ***kwargs*)

Inserts records (as a list of hashes) into a table.

Parameters

- **table** (*str*) – Table name
- **records** (*list*) – Records to be inserted (as a list of hashes)
- **engine** (*str*) – Engine to be used (default: default class engine)
- **key_columns** (*list*) – Set it to specify the names of the key columns for `table`. Set it if you want your statement to be executed with the `on_duplicate_update` flag.
- **on_duplicate_update** (*bool*) – If set, update the records in case of duplicate rows (default: False). If set, you'll need to specify `key_columns` as well.
- **args** – Extra arguments that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)

- **kwargs** – Extra kwargs that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)

Example:

Request:

```
{
    "type": "request",
    "target": "your_host",
    "action": "db.insert",
    "args": {
        "table": "table",
        "engine": "sqlite:///:memory:",
        "records": [
            {
                "id": 1,
                "name": foo
            },

            {
                "id": 2,
                "name": bar
            }
        ]
    }
}
```

select (*query=None*, *table=None*, *filter=None*, *engine=None*, **args*, ***kwargs*)
Returns rows (as a list of hashes) given a query.

Parameters

- **query** (*str*) – SQL to be executed
- **filter** (*dict*) – Query WHERE filter expressed as a dictionary. This approach is preferred over specifying raw SQL in `query` as the latter approach may be prone to SQL injection, unless you need to build some complex SQL logic.
- **table** (*str*) – If you specified a filter instead of a raw query, you'll have to specify the target table
- **engine** (*str*) – Engine to be used (default: default class engine)
- **args** – Extra arguments that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)
- **kwargs** – Extra kwargs that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)

Returns List of hashes representing the result rows.

Examples:

Request:

```
{
    "type": "request",
    "target": "your_host",
    "action": "db.select",
    "args": {
        "engine": "sqlite:///:memory:",
```

(continues on next page)

(continued from previous page)

```
        "query": "SELECT id, name FROM table"
    }
}
```

or:

```
{
    "type": "request",
    "target": "your_host",
    "action": "db.select",
    "args": {
        "engine": "sqlite:///:memory:",
        "table": "table",
        "filter": {"id": 1}
    }
}
```

Response:

```
[
    {
        "id": 1,
        "name": foo
    }
]
```

update (*table*, *records*, *key_columns*, *engine=None*, **args*, ***kwargs*)
Updates records on a table.

Parameters

- **table** (*str*) – Table name
- **records** (*list*) – Records to be updated (as a list of hashes)
- **key_columns** (*list*) – Names of the key columns, used in the WHERE condition
- **engine** (*str*) – Engine to be used (default: default class engine)
- **args** – Extra arguments that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)
- **kwargs** – Extra kwargs that will be passed to `sqlalchemy.create_engine` (see http://docs.sqlalchemy.org/en/latest/core/engines.html)

Example:

Request:

```
{
    "type": "request",
    "target": "your_host",
    "action": "db.update",
    "args": {
        "table": "table",
        "engine": "sqlite:///:memory:",
        "key_columns": ["id"],
        "records": [
            {
```

(continues on next page)

(continued from previous page)

```
                    "id": 1,
                    "name": foo
                },

                {
                    "id": 2,
                    "name": bar
                }
            ]
        }
}
```

2.26 `platypush.plugins.dbus`

class `platypush.plugins.dbus.`**`BusType`**
An enumeration.

class `platypush.plugins.dbus.`**`DbusPlugin`**(***kwargs*)
Plugin to interact with DBus.

Requires:

- **dbus-python** (`pip install dbus-python`)

`__init__`(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

`execute`(*service: str*, *path: str*, *method_name: str*, *args: Optional[list] = None*, *interface: Optional[str] = None*, *bus_type: str = 'session'*)
Execute a method exposed on DBus.

Parameters

- **`service`** – Service/bus name (e.g. `org.platypush.Bus`).
- **`path`** – Object path (e.g. `/MessageService`).
- **`method_name`** – Method name (e.g. `Post`).
- **`args`** – Arguments to be passed to the method, depending on the method signature.
- **`interface`** – Interface name (e.g. `org.platypush.MessageBusInterface`).
- **`bus_type`** – Bus type (supported: `system` and `session` - default: `session`).

Returns Return value of the executed method.

`query`(*service: Optional[str] = None*, *system_bus: bool = True*, *session_bus: bool = True*) → Dict[str, dict]
Query DBus for a specific service or for the full list of services.

Parameters

- **`service`** – Service name (default: None, query all services).
- **`system_bus`** – Query the system bus (default: True).
- **`session_bus`** – Query the session bus (default: True).

Returns A `{service_name -> {properties}}` mapping.

2.27 `platypush.plugins.dropbox`

class `platypush.plugins.dropbox.`**`DropboxPlugin`**(*access_token*, ***kwargs*)

Plugin to manage a Dropbox account and its files and folders.

Requires:

- **dropbox** (`pip install dropbox`)

`__init__`(*access_token*, ***kwargs*)

Parameters **`access_token`** (*str*) – Dropbox API access token. You can get yours by creating an app on https://dropbox.com/developers/apps

`copy`(*from_path: str*, *to_path: str*, *allow_shared_folder=True*, *autorename=False*, *allow_ownership_transfer=False*)

Copy a file or folder to a different location in the user's Dropbox. If the source path is a folder all its contents will be copied.

Parameters

- **`from_path`** – Source path
- **`to_path`** – Destination path
- **`allow_shared_folder`** (*bool*) – If true, `files_copy()` will copy contents in shared folder, otherwise `RelocationError.cant_copy_shared_folder` will be returned if `from_path` contains shared folder. This field is always true for `files_move()`.
- **`autorename`** (*bool*) – If there's a conflict, have the Dropbox server try to autorename the file to avoid the conflict.
- **`allow_ownership_transfer`** (*bool*) – Allow moves by owner even if it would result in an ownership transfer for the content being moved. This does not apply to copies.

`delete`(*path: str*)

Delete the file or folder at a given path. If the path is a folder, all its contents will be deleted too

Parameters **`path`** (*str*) – Path to be removed

`download`(*path: str*, *download_path=None*, *zip=False*)

Download a file or a zipped directory from a user's Dropbox.

Parameters

- **`path`** (*str*) – Dropbox destination path
- **`download_path`** (*str*) – Destination path on the local machine (optional)
- **`zip`** (*bool*) – If set then the content will be downloaded in zip format (default: False)

Return type dict

Returns A dictionary with keys: `('id', 'name', 'parent_shared_folder_id', 'path', 'size', 'encoding', 'content_hash')`. If download_path is set 'file' is also returned. Otherwise 'content' will be returned. If it's a text file then 'content' will contain its string representation, otherwise its base64-encoded representation.

`get_account`(*account_id=None*)

Get the information about a linked Dropbox account

Parameters **`account_id`** (*str*) – account_id. If none is specified then it will retrieve the current user's account id

Returns dict with the following attributes: account_id, name, email, email_verified, disabled, profile_photo_url, team_member_id

`get_metadata`(*path: str*)
Get the metadata of the specified path

Parameters **`path`** (*str*) – Path to the resource

Rtype dict

`get_usage`()
Get the amount of allocated and used remote space

Returns dict

`list`(*folder=''*)
Returns the files in a folder

Parameters **`folder`** (*str*) – Folder name (default: root)

Returns dict

`mkdir`(*path: str*)
Create a folder at a given path.

Parameters **`path`** (*str*) – Folder path

`move`(*from_path: str*, *to_path: str*, *allow_shared_folder=True*, *autorename=False*, *allow_ownership_transfer=False*)
Move a file or folder to a different location in the user's Dropbox. If the source path is a folder all its contents will be moved.

Parameters

- **`from_path`** – Source path
- **`to_path`** – Destination path
- **`allow_shared_folder`** (*bool*) – If true, `files_copy()` will copy contents in shared folder, otherwise `RelocationError.cant_copy_shared_folder` will be returned if `from_path` contains shared folder. This field is always true for `files_move()`.
- **`autorename`** (*bool*) – If there's a conflict, have the Dropbox server try to autorename the file to avoid the conflict.
- **`allow_ownership_transfer`** (*bool*) – Allow moves by owner even if it would result in an ownership transfer for the content being moved. This does not apply to copies.

`restore`(*path: str*, *rev: str*)
Restore a specific revision of a file to the given path.

Parameters

- **`path`** (*str*) – Path to be removed
- **`rev`** (*str*) – Revision ID to be restored

`save`(*path: str*, *url: str*)
Save a specified URL into a file in user's Dropbox. If the given path already exists, the file will be renamed to avoid the conflict (e.g. myfile (1).txt).

Parameters

- **path** – Dropbox destination path
- **url** – URL to download

search(*query: str*, *path=”*, *start=0*, *max_results=100*, *content=False*)

Searches for files and folders. Note: Recent changes may not immediately be reflected in search results due to a short delay in indexing.

Parameters

- **path** (*str*) – The path in the user’s Dropbox to search. Should probably be a folder.
- **query** (*str*) – The string to search for. The search string is split on spaces into multiple tokens. For file name searching, the last token is used for prefix matching (i.e. “bat c” matches “bat cave” but not “batman car”).
- **start** (*long*) – The starting index within the search results (used for paging).
- **max_results** (*long*) – The maximum number of search results to return.
- **content** – Search also in files content (default: False)

Rtype dict

Returns Dictionary with the following fields: `('matches', 'start')`.

upload(*file=None*, *text=None*, *path=’/’*, *overwrite=False*, *autorename=False*)

Create a new file with the contents provided in the request. Do not use this to upload a file larger than 150 MB

Parameters

- **file** (*str*) – File to be uploaded
- **text** (*str*) – Text content to be uploaded
- **path** (*str*) – Path in the user’s Dropbox to save the file.
- **overwrite** (*bool*) – If set, in case of conflict the file will be overwritten (default: append content)
- **autorename** (*bool*) – If there’s a conflict, as determined by `mode`, have the Dropbox server try to autorename the file to avoid conflict.

Rtype dict

Returns Dictionary with the metadata of the uploaded file

2.28 platypush.plugins.esp

class platypush.plugins.esp.**EspPlugin**(*devices: List[Union[platypush.plugins.esp.models.device.Device, dict]] = None*, ***kwargs*)

This plugin allows you to fully control to ESP8266/ESP32 devices connected over WiFi. It uses the WebREPL interface embedded in MicroPython to communicate with the device.

All you need to do is to flash the MicroPython firmware to your device, enable the WebREPL interface, and you can use this plugin to fully control the device remotely without deploying any code to the controller.

- Download the MicroPython firmware for your device.
- Connect your ESP8266/ESP32 via serial/USB port and flash the firmware. For example. using `esptool` and assuming that you have an ESP8266 device connected on `/dev/ttyUSB0`:

```
# Erase the flash memory
esptool.py --port /dev/ttyUSB0 erase_flash
# Flash the firmware
esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash --flash_
↪size=detect 0 esp8266-[version].bin
```

- Access the MicroPython interpreter over serial/USB port. For example, on Linux:

```
picocom /dev/ttyUSB0 -b11520
```

- Configure the WiFi interface:

```
>>> import network
>>> wlan = network.WLAN(network.STA_IF)
>>> wlan.active(True)
>>> wlan.connect('YourSSID', 'YourPassword')
>>> # Print the device IP address
>>> wlan.ifconfig()[0]
>>> '192.168.1.23'
```

- Enable the'WebREPL <https://docs.micropython.org/en/latest/esp8266/quickref.html#webrepl-web-browser-interactive-prompt>'_ interface on the device:

```
>>> import webrepl_setup
```

- Follow the instructions, set a password and reset your device. A websocket service should be available by default on the port 8266 of your ESP8266/ESP32 and it can accept commands sent by platypush.

Requires:

- **websocket-client** (`pip install websocket-client`)

`__init__`(*devices: List[Union[platypush.plugins.esp.models.device.Device, dict]] = None, **kwargs*)

Parameters devices – List of configured device. Pre-configuring devices by name allows you to call the actions directly by device name, instead of specifying `host`, `port` and `password` on each call. It also allows you to interact with PINs by name, if you specified names for them, instead of using the PIN number on your calls. Example configuration:

```
devices:
    - host: 192.168.1.23
      port: 8266        # WebREPL websocket port
      password: pwd1    # Device password
      name: smoke_detector
      pins:
        - number: 14
          name: smoke_sensor
          pwm: False

    - host: 192.168.1.24
      port: 8266
      password: pwd2
      name: smart_switch
      pins:
```

(continues on next page)

(continued from previous page)

```
        - number: 13
          name: relay
          pwm: True
```

adc_read(*pin: int = 0*, ***kwargs*) → int

Read an analog value from a PIN. Note that the ESP8266 only has one analog PIN, accessible on the channel `0`. If you are interested in the actual voltage that is measured then apply `V = Vcc * (value/1024)`, where `Vcc` is the supply voltage provided to the device (usually 3V if connected to the Vcc PIN of an ESP8266).

Parameters

- **pin** – GPIO PIN number (default: 0).
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

Returns A value between `0` and `1024`.

ap_config(*ip: Optional[str] = None*, *netmask: Optional[str] = None*, *gateway: Optional[str] = None*, *dns: Optional[str] = None*, *essid: Optional[str] = None*, *passphrase: Optional[str] = None*, *channel: Optional[int] = None*, *hidden: Optional[bool] = None*, ***kwargs*) → Optional[platypush.message.response.esp.EspWifiConfigResult]

Get or set network properties for the WiFi access point interface. If called with no arguments it will return the configuration of the interface.

Parameters

- **ip** – IP address.
- **netmask** – Netmask.
- **gateway** – Default gateway address.
- **dns** – Default DNS address.
- **essid** – ESSID of the access point.
- **passphrase** – Password/passphrase.
- **channel** – WiFi channel.
- **hidden** – Whether the network is hidden.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

ap_disable(***kwargs*)

Disable the device WiFi access point interface.

Parameters kwargs – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

ap_enable(***kwargs*)

Enable the device WiFi access point interface.

Parameters kwargs – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

chdir(*directory: str*, ***kwargs*)

Move to the specified directory.

Parameters

- **`directory`** – Directory name.
- **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.execute()`*.

`close`(*device: Optional[str] = None*, *host: Optional[str] = None*, *port: int = 8266*)
Close an active connection to a device. :param device: Configured device name. Either `device` or `host` and `port` must be provided. :param host: ESP host. :param port: ESP port.

`connect`(*device: Optional[str] = None*, *host: Optional[str] = None*, *port: int = 8266*, *password: Optional[str] = None*, *timeout: Optional[float] = 10.0*)
Open a connection to an ESP device.

Parameters

- **`device`** – Configured device name. Either `device` or `host` and `port` must be provided.
- **`host`** – ESP host.
- **`port`** – ESP port (default: 8266).
- **`password`** – ESP WebREPL password.
- **`timeout`** – Connection timeout (default: 10 seconds).

`db_get`(*dbfile: str*, *key: str*, ***kwargs*) → Any
Set a value on an internal B-Tree file database.

Parameters

- **`dbfile`** – Database file name.
- **`key`** – Key to set.
- **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.execute()`*.

Returns Whichever value is stored as output, or null if the key doesn't exist.

`db_items`(*dbfile: str*, ***kwargs*) → Dict[str, str]
Get a key->value mapping of the items stored in a B-Tree file database.

Parameters

- **`dbfile`** – Database file name.
- **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.execute()`*.

Returns Whichever value is stored as output, or null if the key doesn't exist.

`db_keys`(*dbfile: str*, ***kwargs*) → List[str]
Get the list of keys stored on an internal B-Tree file database.

Parameters

- **`dbfile`** – Database file name.
- **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.execute()`*.

`db_set`(*dbfile: str*, *key: str*, *value: Any*, ***kwargs*)
Set a value on an internal B-Tree file database.

Parameters

- **`dbfile`** – Database file name.

- **key** – Key to set.
- **value** – Value to set.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

db_values (*dbfile: str*, ***kwargs*) → List[str]
Get the list of item values stored on an internal B-Tree file database.

Parameters

- **dbfile** – Database file name.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

deep_sleep (*seconds: Optional[float] = None*, ***kwargs*)
Stops execution in an attempt to enter a low power state. A deepsleep may not retain RAM or any other state of the system (for example peripherals or network interfaces). Upon wake execution is resumed from the main script, similar to a hard or power-on reset.

Parameters

- **seconds** – Sleep seconds (default: sleep until there are some PIN/RTC events to process)
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

dht11_get_humidity (*pin: Union[int, str]*, ***kwargs*) → float
Get the humidity value in percentage (0-100) from a connected DHT11 sensor.

Parameters

- **pin** – GPIO PIN number or configured name where the sensor is connected.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

dht11_get_temperature (*pin: Union[int, str]*, ***kwargs*) → float
Get the temperature value in Celsius from a connected DHT11 sensor.

Parameters

- **pin** – GPIO PIN number or configured name where the sensor is connected.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

dht22_get_humidity (*pin: Union[int, str]*, ***kwargs*) → float
Get the humidity value in percentage (0-100) from a connected DHT22 sensor.

Parameters

- **pin** – GPIO PIN number or configured name where the sensor is connected.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

dht22_get_temperature (*pin: Union[int, str]*, ***kwargs*) → float
Get the temperature value in Celsius from a connected DHT22 sensor.

Parameters

- **pin** – GPIO PIN number or configured name where the sensor is connected.

- **kwargs** – Parameters to pass to `platypush.plugins.esp.EspPlugin.execute()`.

disable_irq(***kwargs*)

Disable interrupt requests.

Parameters kwargs – Parameters to pass to `platypush.plugins.esp.EspPlugin.execute()`.

enable_irq(***kwargs*)

Enable interrupt requests.

Parameters kwargs – Parameters to pass to `platypush.plugins.esp.EspPlugin.execute()`.

execute(*code: str, device: Optional[str] = None, host: Optional[str] = None, port: int = 8266, password: Optional[str] = None, conn_timeout: Optional[float] = 10.0, recv_timeout: Optional[float] = 30.0, wait_response: bool = True, **kwargs*) → platypush.message.response.Response

Run raw Python code on the ESP device.

Parameters

- **code** – Snippet of code to run.
- **device** – Configured device name. Either `device` or `host` and `port` must be provided.
- **host** – ESP host.
- **port** – ESP port (default: 8266).
- **password** – ESP WebREPL password.
- **conn_timeout** – Connection timeout (default: 10 seconds).
- **recv_timeout** – Response receive timeout (default: 30 seconds).
- **wait_response** – Wait for the response from the device (default: True)

Returns The response returned by the Micropython interpreter, as a string.

file_download(*source: str, destination: str, timeout: Optional[float] = 60.0, **kwargs*)

Download a file from the board to the local machine.

Parameters

- **source** – Name or path of the file to download from the device.
- **destination** – Target directory or path on the local machine.
- **timeout** – File transfer timeout (default: one minute).
- **kwargs** – Parameters to pass to `platypush.plugins.esp.EspPlugin.execute()`.

file_get(*file: str, binary: bool = False, timeout: Optional[float] = 60.0, **kwargs*) → str

Get the content of a file on the board.

Parameters

- **file** – File name/path to get from the device.
- **binary** – If True, then the base64-encoded content of the file will be returned.
- **timeout** – File transfer timeout (default: one minute).

- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.connect()*.

file_upload(*source: str*, *destination: Optional[str] = None*, *timeout: Optional[float] = 60.0*, ***kwargs*)
Upload a file to the board.

Parameters

- **source** – Path of the local file to copy.
- **destination** – Target file name (default: a filename will be created under the board's root folder with the same name as the source file).
- **timeout** – File transfer timeout (default: one minute).
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.connect()*.

get_freq(***kwargs*) → int
Get the frequency of the device in Hz.

Parameters kwargs – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

i2c_close(***kwargs*)
Turn off an I2C bus.

Parameters kwargs – Parameters to pass to *platypush.plugins.esp.EspPlugin.i2c_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

i2c_open(*scl: Optional[int] = None*, *sda: Optional[int] = None*, *id: int = -1*, *baudrate: int = 400000*, ***kwargs*)
Open a connection to an I2C (or "two-wire") port.

Parameters

- **scl** – PIN number for the SCL (serial clock) line.
- **sda** – PIN number for the SDA (serial data) line.
- **id** – The default value of -1 selects a software implementation of I2C which can work (in most cases) with arbitrary pins for SCL and SDA. If id is -1 then scl and sda must be specified. Other allowed values for id depend on the particular port/board, and specifying scl and sda may or may not be required or allowed in this case.
- **baudrate** – Port frequency/clock rate (default: 400 kHz).
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

i2c_read(*address: int*, *size: int*, ***kwargs*) → str
Read data from the I2C bus.

Parameters

- **address** – I2C address.
- **size** – Number of bytes to read.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.i2c_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

Returns String representation of the read bytes, or base64-encoded representation if the data can't be decoded to a string.

i2c_scan(***kwargs*) → List[int]

Scan for device addresses on the I2C bus.

Parameters kwargs – Parameters to pass to *platypush.plugins.esp.EspPlugin.i2c_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

Returns List of 7-bit addresses.

i2c_start(***kwargs*)

Generate a START condition on an I2C bus.

Parameters kwargs – Parameters to pass to *platypush.plugins.esp.EspPlugin.i2c_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

Returns String representation of the read bytes, or base64-encoded representation if the data can't be decoded to a string.

i2c_stop(***kwargs*)

Generate a STOP condition on an I2C bus.

Parameters kwargs – Parameters to pass to *platypush.plugins.esp.EspPlugin.i2c_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

Returns String representation of the read bytes, or base64-encoded representation if the data can't be decoded to a string.

i2c_write(*address: int*, *data: str*, *binary: bool = False*, ***kwargs*)

Write data to the I2C bus.

Parameters

- **address** – I2C address.
- **data** – Data to be sent.
- **binary** – By default data will be treated as a string. Set binary to True if it should instead be treated as a base64-encoded binary string to be decoded before being sent.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.i2c_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

Returns String representation of the read bytes, or base64-encoded representation if the data can't be decoded to a string.

listdir(*directory: str = '/'*, ***kwargs*) → List[str]

List the content of a directory.

Parameters

- **directory** – Directory name (default: root).
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

mkdir(*directory: str*, ***kwargs*)

Create a directory.

Parameters

- **directory** – Directory name.

- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pin_off (*pin: Union[int, str], pull_up: bool = False, **kwargs*)
Set the specified PIN to LOW.

Parameters

- **pin** – GPIO PIN number.
- **pull_up** – Set to True if the PIN has a (weak) pull-up resistor attached.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pin_on (*pin: Union[int, str], pull_up: bool = False, **kwargs*)
Set the specified PIN to HIGH.

Parameters

- **pin** – GPIO PIN number or configured name.
- **pull_up** – Set to True if the PIN has a (weak) pull-up resistor attached.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pin_read (*pin: Union[int, str], out: bool = False, pull_up: bool = False, **kwargs*) → bool
Get the ON/OFF value of a PIN.

Parameters

- **pin** – GPIO PIN number or configured name.
- **out** – Treat the PIN as an output PIN - e.g. if you usually write to it and now want to read the value. If not set, then the PIN will be treated as an input PIN.
- **pull_up** – Set to True if the PIN has a (weak) pull-up resistor attached.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pin_toggle (*pin: Union[int, str], pull_up: bool = False, **kwargs*)
Toggle a PIN state - to HIGH if LOW, to LOW if HIGH.

Parameters

- **pin** – GPIO PIN number or configured name.
- **pull_up** – Set to True if the PIN has a (weak) pull-up resistor attached.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pwm_duty (*pin: Union[int, str], duty: Optional[int] = None, **kwargs*) → Optional[int]
Get/set the duty cycle of a PWM PIN.

Parameters

- **pin** – GPIO PIN number or configured name.
- **duty** – Optional duty value to set.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pwm_freq (*pin: Union[int, str], freq: Optional[int] = None, **kwargs*) → Optional[int]
Get/set the frequency of a PWM PIN.

Parameters

- **pin** – GPIO PIN number or configured name.
- **freq** – If set, set the frequency for the PIN in Hz.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pwm_off (*pin: Union[int, str], **kwargs*)
Turn off a PWM PIN.

Parameters

- **pin** – GPIO PIN number or configured name.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

pwm_on (*pin: Union[int, str], freq: Optional[int] = None, duty: Optional[int] = None, **kwargs*)
Set the specified PIN to HIGH.

Parameters

- **pin** – GPIO PIN number or configured name.
- **freq** – PWM PIN frequency.
- **duty** – PWM PIN duty cycle.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

remove (*file: str, **kwargs*)
Remove a file.

Parameters

- **file** – File name/path.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

rename (*name: str, new_name: str, **kwargs*)
Rename a file or directory.

Parameters

- **name** – Current resource name.
- **new_name** – New resource name.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

reset (***kwargs*)
Perform a device reset, similar to the user pushing the RESET button. :param kwargs: Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

rmdir (*directory: str, **kwargs*)
Remove a directory.

Parameters

- **directory** – Directory name.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

set_freq(*freq: int*, ***kwargs*)
Set the frequency of the device. :param freq: New frequency in Hz. :param kwargs: Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

set_ntp_time(***kwargs*)
Set the device time using an NTP server.

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

set_password(*new_password: str*, ***kwargs*)
Change the WebREPL password for the device.

Parameters

- **new_password** – New password.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

sleep(*seconds: float*, ***kwargs*)
Perform a software sleep (i.e. `time.sleep()`).

Parameters

- **seconds** – Sleep seconds.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

soft_reset(***kwargs*)
Performs a soft reset of the interpreter, deleting all Python objects and resetting the Python heap. :param kwargs: Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

soft_sleep(*seconds: Optional[float] = None*, ***kwargs*)
Stops execution in an attempt to enter a low power state. A light-sleep has full RAM and state retention. Upon wake execution is resumed from the point where the sleep was requested, with all subsystems operational.

Parameters

- **seconds** – Sleep seconds (default: sleep until there are some PIN/RTC events to process)
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

spi_close(***kwargs*)
Turn off an SPI bus.

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.spi_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

spi_open(*id=1*, *baudrate: int = 1000000*, *polarity: int = 0*, *phase: int = 0*, *bits: int = 8*, *sck: Optional[int] = None*, *mosi: Optional[int] = None*, *miso: Optional[int] = None*, ***kwargs*)
Open a connection to an SPI port. Note that `sck`, `mosi` and `miso` parameters are only allowed if you're setting up a software managed SPI connection. If you're using the hardware SPI implementation then those PINs are pre-defined depending on the model of your board.

Parameters

- **`id`** – Values of id depend on a particular port and its hardware. Values 0, 1, etc. are commonly used to select hardware SPI block #0, #1, etc. Value -1 can be used for bit-banging (software) implementation of SPI (if supported by a port).
- **`baudrate`** – Port baudrate/SCK clock rate (default: 1 MHz).
- **`polarity`** – It can be 0 or 1, and is the level the idle clock line sits at.
- **`phase`** – It can be 0 or 1 to sample data on the first or second clock edge respectively.
- **`bits`** – Number of bits per character. It can be 7, 8 or 9.
- **`sck`** – SCK PIN number.
- **`mosi`** – MOSI PIN number.
- **`miso`** – MISO PIN number.
- **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.execute()`*.

`spi_read`(*size: int*, ***kwargs*) → str

Read from an SPI bus.

Parameters

- **`size`** – Number of bytes to read.
- **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.spi_open()`* and *`platypush.plugins.esp.EspPlugin.execute()`*.

Returns String representation of the read bytes, or base64-encoded representation if the data can't be decoded to a string.

`spi_write`(*data: str*, *binary: bool = False*, ***kwargs*)

Write data to an SPI bus.

Parameters

- **`data`** – Data to be written.
- **`binary`** – By default data will be treated as a string. Set binary to True if it should instead be treated as a base64-encoded binary string to be decoded before being sent.
- **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.spi_open()`* and *`platypush.plugins.esp.EspPlugin.execute()`*.

`uart_close`(***kwargs*)

Turn off the UART bus.

Parameters **`kwargs`** – Parameters to pass to *`platypush.plugins.esp.EspPlugin.uart_open()`* and *`platypush.plugins.esp.EspPlugin.execute()`*.

`uart_open`(*id=1*, *baudrate: Optional[int] = 9600*, *bits: Optional[int] = 8*, *parity: Optional[int] = None*, *stop: int = 1*, *tx_pin: Optional[int] = None*, *rx_pin: Optional[int] = None*, *timeout: Optional[float] = None*, *timeout_char: Optional[float] = None*, ***kwargs*)

Open a connection to a UART port.

Parameters

- **`id`** – Bus ID (default: 1).
- **`baudrate`** – Port baudrate (default: 9600).

- **bits** – Number of bits per character. It can be 7, 8 or 9.
- **parity** – Parity configuration. It can be None (no parity), 0 (even) or 1 (odd).
- **stop** – Number of stop bits. It can be 1 or 2.
- **tx_pin** – Specify the TX PIN to use.
- **rx_pin** – Specify the RX PIN to use.
- **timeout** – Specify the time to wait for the first character in seconds.
- **timeout_char** – Specify the time to wait between characters in seconds.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

uart_read(*size: Optional[int] = None*, ***kwargs*) → str

Read from a UART interface.

Parameters

- **size** – Number of bytes to read (default: read all available characters).
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.uart_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

Returns String representation of the read bytes, or base64-encoded representation if the data can't be decoded to a string.

uart_readline(***kwargs*) → str

Read a line (any character until newline is found) from a UART interface.

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.uart_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

Returns String representation of the read bytes, or base64-encoded representation if the data can't be decoded to a string.

uart_send_break(***kwargs*)

Send a break condition to a UART bus. This drives the bus low for a duration longer than required for a normal transmission of a character.

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.uart_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

uart_write(*data: str*, *binary: bool = False*, ***kwargs*)

Write data to the UART bus.

Parameters

- **data** – Data to be written.
- **binary** – By default data will be treated as a string. Set binary to True if it should instead be treated as a base64-encoded binary string to be decoded before being sent.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.uart_open()* and *platypush.plugins.esp.EspPlugin.execute()*.

unique_id(***kwargs*) → str

Get the unique ID of the device. t will vary from a board/SoC instance to another, if underlying hardware allows. Length varies by hardware (so use substring of a full value if you expect a short ID). In some MicroPython ports, ID corresponds to the network MAC address..

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

urandom(*size: int = 1*, ***kwargs*) → List[int]
Get randomly generated bytes.

Parameters

- **size** – Number of random bytes to be generated (default: 1).
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

wifi_config(*ip: Optional[str] = None*, *netmask: Optional[str] = None*, *gateway: Optional[str] = None*, *dns: Optional[str] = None*, ***kwargs*) → Optional[platypush.message.response.esp.EspWifiConfigResult]
Get or set network properties for the WiFi interface. If called with no arguments it will return the configuration of the interface.

Parameters

- **ip** – IP address.
- **netmask** – Netmask.
- **gateway** – Default gateway address.
- **dns** – Default DNS address.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

wifi_connect(*essid: str*, *passphrase: str*, ***kwargs*)
Connect the device WiFi interface to the specified access point.

Parameters

- **essid** – WiFi ESSID.
- **passphrase** – WiFi passphrase.
- **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

wifi_disable(***kwargs*)
Disable the device WiFi interface.

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

wifi_disconnect(***kwargs*)
Disconnect from the currently connected WiFi network

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

wifi_enable(***kwargs*)
Enable the device WiFi interface.

Parameters **kwargs** – Parameters to pass to *platypush.plugins.esp.EspPlugin.execute()*.

wifi_scan(***kwargs*) → List[platypush.message.response.esp.EspWifiScanResult]
Scan the available networks.

Parameters **`kwargs`** – Parameters to pass to `platypush.plugins.esp.EspPlugin.execute()`.

2.29 `platypush.plugins.ffmpeg`

class `platypush.plugins.ffmpeg.`**`FfmpegPlugin`**(*ffmpeg_cmd: str = 'ffmpeg', ffprobe_cmd: str = 'ffprobe', **kwargs*)

Generic FFmpeg plugin to interact with media files and devices.

Requires:

- **ffmpeg-python** (`pip install ffmpeg-python`)
- The **ffmpeg** package installed on the system.

`__init__`(*ffmpeg_cmd: str = 'ffmpeg', ffprobe_cmd: str = 'ffprobe', **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

`info`(*filename: str, **kwargs*) → dict

Get the information of a media file.

Parameters **`filename`** – Path to the media file.

Returns

Media file information. Example:

```
{
  "streams": [
      {
          "index": 0,
          "codec_name": "h264",
          "codec_long_name": "H.264 / AVC / MPEG-4 AVC / MPEG-4␣
↪part 10",
          "profile": "High 4:2:2",
          "codec_type": "video",
          "codec_time_base": "1/60",
          "codec_tag_string": "[0][0][0][0]",
          "codec_tag": "0x0000",
          "width": 640,
          "height": 480,
          "coded_width": 640,
          "coded_height": 480,
          "closed_captions": 0,
          "has_b_frames": 2,
          "pix_fmt": "yuv422p",
          "level": 30,
          "chroma_location": "left",
          "field_order": "progressive",
          "refs": 1,
          "is_avc": "true",
          "nal_length_size": "4",
          "r_frame_rate": "30/1",
          "avg_frame_rate": "30/1",
          "time_base": "1/1000",
          "start_pts": 0,
          "start_time": "0.000000",
          "bits_per_raw_sample": "8",
```

(continues on next page)

(continued from previous page)

```
                "disposition": {
                    "default": 1,
                    "dub": 0,
                    "original": 0,
                    "comment": 0,
                    "lyrics": 0,
                    "karaoke": 0,
                    "forced": 0,
                    "hearing_impaired": 0,
                    "visual_impaired": 0,
                    "clean_effects": 0,
                    "attached_pic": 0,
                    "timed_thumbnails": 0
                },
                "tags": {
                    "ENCODER": "Lavc58.91.100 libx264"
                }
            }
        ],
        "format": {
            "filename": "./output.mkv",
            "nb_streams": 1,
            "nb_programs": 0,
            "format_name": "matroska,webm",
            "format_long_name": "Matroska / WebM",
            "start_time": "0.000000",
            "size": "786432",
            "probe_score": 100,
            "tags": {
                "ENCODER": "Lavf58.45.100"
            }
        }
}
```

2.30 `platypush.plugins.file`

class `platypush.plugins.file.`**`FilePlugin`**(***kwargs*)

A plugin for general-purpose file methods

`append`(*file: str*, *content*)

Append content to a specified (text) file.

Parameters

- **`file`** – Path of the file.
- **`content`** – Content to write.

`chmod`(*file: str*, *mode*)

Change the mode/permissions of a file.

Parameters

- **`file`** – File name/path.
- **`mode`** – New file permissions.

`getsize` (*file*)
Get the size of the specified file in bytes.

Parameters **`file`** – File path.

`home` () → str
Returns the current user's home directory.

`link` (*file: str*, *target: str*, *symbolic=True*)
Create a link to a file.

Parameters

- **`file`** – File to symlink.
- **`target`** – Symlink path.
- **`symbolic`** – If True, then the target link will be a symbolic link. Otherwise, it will be a hard link (default: symbolic).

`list` (*path: str = '/'*) → List[Dict[str, str]]
List a file or all the files in a directory.

Parameters **`path`** – File or directory (default: root directory).

Returns List of files in the specified path, or absolute path of the specified path if `path` is a file and it exists. Each item will contain the fields `type` (`file` or `directory`) and `path`.

`mkdir` (*directory: str*, *exist_ok=True*, *parents=True*, *mode=493*)
Create a directory.

Parameters

- **`directory`** – Directory name/path.
- **`exist_ok`** – If set and the directory already exist the method will not return an error (default: True).
- **`parents`** – If set and any of the parent directories in the path don't exist they will be created (analogous to mkdir -p) (default: True).
- **`mode`** – Access mode (default: 0755).

`read` (*file: str*)
Read and return the content of a (text) file.

Parameters **`file`** – Path of the file.

`rename` (*file: str*, *name: str*)
Rename/move a file.

Parameters

- **`file`** – File to rename.
- **`name`** – New file name.

`rmdir` (*directory: str*)
Remove a directory. The directory must be empty.

Parameters **`directory`** – Directory name/path.

`touch` (*file: str*, *mode=420*)
Create/touch a file.

Parameters

- **file** – File name/path.
- **mode** – File permissions (default: 0644).

unlink (*file: str*)
Remove a file or symbolic link.

Parameters **file** – File/link to remove.

write (*file: str*, *content: str*)
Writes content to a specified (text) file. Previous content will be truncated.

Parameters

- **file** – Path of the file.
- **content** – Content to write.

2.31 platypush.plugins.foursquare

class platypush.plugins.foursquare.**FoursquarePlugin** (*access_token: str*, ***kwargs*)
Plugin to interact with the Foursquare Places API.

In order to enable the Foursquare API on your account you need to:

- Create a new app on the Foursquare developers website.
- Copy the `client_id` and `client_secret`.
- **Add a redirect URL. It must point to a valid IP/hostname with a web server running, even if it runs** locally. You can also use the local URL of the platypush web server - e.g. http://192.168.1.2:8008/.
- **Open the following URL: `https://foursquare.com/oauth2/authenticate?client_id=CLIENT_ID&r`** Replace `CLIENT_ID` and `REDIRECT_URI` with the parameters from your app.
- **Allow the application. You will be redirected to the URL you provided. Copy the `access_token` provided in** the URL.

__init__ (*access_token: str*, ***kwargs*)

Parameters **access_token** –

checkin (*venue_id: str*, *latitude: Optional[float] = None*, *longitude: Optional[float] = None*, *altitude: Optional[float] = None*, *latlng_accuracy: Optional[float] = None*, *altitude_accuracy: Optional[float] = None*, *shout: Optional[str] = None*, *broadcast: Optional[List[str]] = None*) → Dict[str, Any]
Create a new check-in.

Parameters

- **venue_id** – ID of the venue to check-in.
- **latitude** – Check-in latitude.
- **longitude** – Check-in longitude.
- **altitude** – Check-in altitude.
- **latlng_accuracy** – Latitude/longitude accuracy in meters.
- **altitude_accuracy** – Altitude accuracy in meters.
- **shout** – Add a custom message to the check-in.

- **`broadcast`** – List of Visibility/share types of the check-in. Default: `public`. Possible values are:
  - `private`
  - `public`
  - `followers`
  - `facebook`
  - `twitter`

Returns Foursquare API response.

`explore`(*latitude: Optional[float] = None*, *longitude: Optional[float] = None*, *altitude: Optional[float] = None*, *latlng_accuracy: Optional[float] = None*, *altitude_accuracy: Optional[float] = None*, *section: Optional[str] = None*, *near: Optional[str] = None*, *query: Optional[str] = None*, *limit: Optional[int] = None*, *categories: Optional[List[str]] = None*, *radius: Optional[int] = None*, *open_now: bool = True*, *sort_by_distance: Optional[bool] = None*, *sort_by_popularity: Optional[bool] = None*, *price: Optional[List[int]] = None*, *saved: Optional[bool] = None*) → List[Dict[str, Any]]

Explore venues around a location.

Parameters

- **`latitude`** – Search near this latitude. Note either `latitude, longitude` or `near` should be provided.
- **`longitude`** – Search near this latitude. Note either `latitude, longitude` or `near` should be provided.
- **`near`** – Search near this place (e.g. "Chicago, IL" or "Amsterdam, NL"). Note either `latitude, longitude` or `near` should be provided.
- **`altitude`** – Search near this altitude in meters.
- **`latlng_accuracy`** – Latitude/longitude accuracy in meters.
- **`altitude_accuracy`** – Altitude accuracy in meters.
- **`section`** – Section to search. Supported values:
  - food
  - drinks
  - coffee
  - shops
  - arts
  - outdoors
  - sights
  - trending
  - nextVenues
- **`query`** – Search query (e.g. "coffee shops" or "restaurants"). The parameter has no effect if `section` is specified.
- **`limit`** – Maximum number of results.
- **`categories`** – List of category IDs to be searched.

- **radius** – Search radius in meters.
- **open_now** – Filter by open/not open now.
- **sort_by_distance** – Sort by distance.
- **sort_by_popularity** – Sort by popularity
- **price** – Price ranges, within the range `[1,2,3,4]`.
- **saved** – Filter by saved/unsaved venues.

Returns A list of venues, as returned by the Foursquare API.

get_checkins() → List[Dict[str, Any]]
Get the list of check-ins of the current user. :return: A list of checkins, as returned by the Foursquare API.

managed() → List[Dict[str, Any]]
Get the list of venues managed by the user. :return: A list of venues, as returned by the Foursquare API.

search(*latitude: Optional[float] = None, longitude: Optional[float] = None, altitude: Optional[float] = None, latlng_accuracy: Optional[float] = None, altitude_accuracy: Optional[float] = None, near: Optional[str] = None, query: Optional[str] = None, limit: Optional[int] = None, url: Optional[int] = None, categories: Optional[List[str]] = None, radius: Optional[int] = None, sw: Union[Tuple[float], List[float], None] = None, ne: Union[Tuple[float], List[float], None] = None*) → List[Dict[str, Any]]
Search for venues.

Parameters

- **latitude** – Search near this latitude. Note either `latitude, longitude` or `near` should be provided.
- **longitude** – Search near this latitude. Note either `latitude, longitude` or `near` should be provided.
- **near** – Search near this place (e.g. "Chicago, IL" or "Amsterdam, NL"). Note either `latitude, longitude` or `near` should be provided.
- **altitude** – Search near this altitude in meters.
- **latlng_accuracy** – Latitude/longitude accuracy in meters.
- **altitude_accuracy** – Altitude accuracy in meters.
- **query** – Search query (e.g. "coffee shops" or "restaurants").
- **limit** – Maximum number of results.
- **url** – Venue URL to search.
- **categories** – List of category IDs to be searched.
- **radius** – Search radius in meters.
- **sw** – South/west boundary box as a `[latitude, longitude]` pair.
- **ne** – North/east boundary box as a `[latitude, longitude]` pair.

Returns A list of venues, as returned by the Foursquare API.

stats(*venue_id: str, start_at: Union[int, float, datetime.datetime, str], end_at: Union[int, float, datetime.datetime, str]*) → List[Dict[str, Any]]
Get the stats about a venue over a time range. The user must be a manager of that venue.

Parameters

- **venue_id** – Venue ID.

- **start_at** – Stats start time. Can be a UNIX timestamp, a datetime object or an ISO format datetime.
- **end_at** – Stats end time. Can be a UNIX timestamp, a datetime object or an ISO format datetime.

Returns A list of venues, as returned by the Foursquare API.

time_series(*venue_id: Union[str, List[str]], start_at: Union[int, float, datetime.datetime, str], end_at: Union[int, float, datetime.datetime, str]*) → List[Dict[str, Any]]

Get the visitors stats about one or multiple venues over a time range. The user must be a manager of those venues.

Parameters

- **venue_id** – Venue ID or list of IDs to get the stats for.
- **start_at** – Stats start time. Can be a UNIX timestamp, a datetime object or an ISO format datetime.
- **end_at** – Stats end time. Can be a UNIX timestamp, a datetime object or an ISO format datetime.

Returns A list of venues, as returned by the Foursquare API.

trending(*latitude: Optional[float] = None*, *longitude: Optional[float] = None*, *near: Optional[str] = None*, *limit: Optional[int] = None*, *radius: Optional[int] = None*) → List[Dict[str, Any]]

Get the trending venues around a location.

Parameters

- **latitude** – Search near this latitude. Note either `latitude, longitude` or `near` should be provided.
- **longitude** – Search near this latitude. Note either `latitude, longitude` or `near` should be provided.
- **near** – Search near this place (e.g. "Chicago, IL" or "Amsterdam, NL"). Note either `latitude, longitude` or `near` should be provided.
- **limit** – Maximum number of results.
- **radius** – Search radius in meters.

Returns A list of venues, as returned by the Foursquare API.

2.32 platypush.plugins.google

class platypush.plugins.google.**GooglePlugin**(*scopes=None*, **args*, ***kwargs*)

Executes calls to the Google APIs using the google-api-python-client. This class is extended by `GoogleMailPlugin`, `GoogleCalendarPlugin` etc. In order to use Google services (like GMail, Maps, Calendar etc.) with your account you need to:

1. Create your Google application, if you don't have one already, on the developers console, https://console.developers.google.com

2. Click on "Credentials", then "Create credentials" -> "OAuth client ID"

3 Select "Other", enter whichever description you like, and create

4. Click on the "Download JSON" icon next to your newly created client ID

5. Generate a credentials file for the needed scope:

```
python -m platypush.plugins.google.credentials 'https://www.
↪googleapis.com/auth/gmail.compose' ~/client_secret.json
```

Requires:

- **google-api-python-client** (`pip install google-api-python-client`)
- **oauth2client** (`pip install oauth2client`)

`__init__`(*scopes=None*, **args*, ***kwargs*)
Initialized the Google plugin with the required scopes.

Parameters **`scopes`** (*`list`*) – List of required scopes

2.33 `platypush.plugins.google.calendar`

class `platypush.plugins.google.calendar.`**`GoogleCalendarPlugin`**(**args*, ***kwargs*)
Google calendar plugin

`__init__`(**args*, ***kwargs*)
Initialized the Google plugin with the required scopes.

Parameters **`scopes`** (*`list`*) – List of required scopes

`get_upcoming_events`(*max_results=10*)
Get the upcoming events. See *`get_upcoming_events()`*.

2.34 `platypush.plugins.google.drive`

class `platypush.plugins.google.drive.`**`GoogleDrivePlugin`**(**args*, ***kwargs*)
Google Drive plugin.

`__init__`(**args*, ***kwargs*)
Initialized the Google plugin with the required scopes.

Parameters **`scopes`** (*`list`*) – List of required scopes

`copy`(*file_id: str*) → platypush.message.response.google.drive.GoogleDriveFile
Create a copy of a file. :param file_id: File ID.

`create`(*name: str*, *description: Optional[str] = None*, *mime_type: Optional[str] = None*, *parents: Optional[List[str]] = None*, *starred: bool = False*) → platypush.message.response.google.drive.GoogleDriveFile
Create a file.

Parameters

- **`name`** – File name.
- **`description`** – File description.
- **`mime_type`** – File MIME type.
- **`parents`** – List of folder IDs that will contain the file (default: drive root).
- **`starred`** – If True then the file will be marked as starred.

`delete`(*file_id: str*)
Delete a file from Google Drive. :param file_id: File ID.

download (*file_id: str*, *path: str*) → str

Download a Google Drive file locally.

Parameters

- **file_id** – Path of the file to upload.
- **path** – Path of the file to upload.

Returns The local file path.

empty_trash ()

Empty the Drive bin.

files (*filter: Optional[str] = None*, *folder_id: Optional[str] = None*, *limit: Optional[int] = 100*, *drive_id: Optional[str] = None*, *spaces: Union[str, List[str], None] = None*, *order_by: Union[str, List[str], None] = None*) → Union[platypush.message.response.google.drive.GoogleDriveFile, List[platypush.message.response.google.drive.GoogleDriveFile]]

Get the list of files.

Parameters

- **filter** – Optional filter (default: None). See Google Drive API docs for the supported syntax.
- **folder_id** – Drive folder ID to search (default: get all files).
- **limit** – Maximum number of entries to be retrieves (default: 100).
- **drive_id** – Shared drive ID to search (default: None).
- **spaces** – Drive spaces to search. Supported values:
  - `drive`
  - `appDataFolder`
  - `photos`
- **order_by** – Order the results by a specific attribute or list of attributes (default: None). Supported attributes:
  - `createdTime`
  - `folder`
  - `modifiedByMeTime`
  - `modifiedTime`
  - `name`
  - `name_natural`
  - `quotaBytesUsed`
  - `recency`
  - `sharedWithMeTime`
  - `starred`
  - `viewedByMeTime`

  Attributes will be sorted in ascending order by default. You can change that by by appending "`desc`" separated by a space to the attribute you want in descending order - e.g. `["folder", "createdTime desc", "modifiedTime desc"]`.

get (*file_id: str*)
Get the information of a file on the Drive by file ID. :param file_id: File ID.

update (*file_id: str, name: Optional[str] = None, description: Optional[str] = None, add_parents: Optional[List[str]] = None, remove_parents: Optional[List[str]] = None, mime_type: Optional[str] = None, starred: bool = None, trashed: bool = None*) → platypush.message.response.google.drive.GoogleDriveFile
Update the metadata or the content of a file.

Parameters

- **file_id** – File ID.
- **name** – Set the file name.
- **description** – Set the file description.
- **add_parents** – Add the file to these parent folder IDs.
- **remove_parents** – Remove the file from these parent folder IDs.
- **mime_type** – Set the file MIME type.
- **starred** – Change the starred flag.
- **trashed** – Move/remove from trash.

upload (*path: str, mime_type: Optional[str] = None, name: Optional[str] = None, description: Optional[str] = None, parents: Optional[List[str]] = None, starred: bool = False, target_mime_type: Optional[str] = None*) → platypush.message.response.google.drive.GoogleDriveFile
Upload a file to Google Drive.

Parameters

- **path** – Path of the file to upload.
- **mime_type** – MIME type of the source file (e.g. "`image/jpeg`").
- **name** – Name of the target file. Default: same name as the source file.
- **description** – File description.
- **parents** – List of folder IDs that will contain the file (default: drive root).
- **starred** – If True, then the uploaded file will be marked as starred by the user.
- **target_mime_type** – Target MIME type. Useful if you want to e.g. import a CSV file as a Google Sheet (use "`application/vnd.google-apps.spreadsheet`), or an ODT file to a Google Doc (use "`application/vnd.google-apps.document`). See the official documentation for a complete list of supported types.

2.35 platypush.plugins.google.fit

class `platypush.plugins.google.fit.`**GoogleFitPlugin** (*user_id='me', *args, **kwargs*)
Google Fit plugin

__init__ (*user_id='me', *args, **kwargs*)

Parameters **user_id** (*str or int*) – Default Google user_id (default: 'me', default configured account user)

get_data (*data_source_id*, *user_id=None*, *limit=None*)
Get raw data for the specified data_source_id

Parameters **data_source_id** (*str*) – Data source ID, see *get_data_sources*

get_data_sources (*user_id=None*)
Get the available data sources for the specified user_id

2.36 platypush.plugins.google.mail

class platypush.plugins.google.mail.**GoogleMailPlugin** (**args*, ***kwargs*)
GMail plugin. It allows you to programmatically compose and (TODO) get emails

__init__ (**args*, ***kwargs*)
Initialized the Google plugin with the required scopes.

Parameters **scopes** (*list*) – List of required scopes

compose (*sender*, *to*, *subject*, *body*, *files=None*)
Compose a message.

Parameters

- **sender** (*str*) – Sender email/name
- **to** (*str*) – Recipient email or comma-separated list of recipient emails
- **subject** (*str*) – Email subject
- **body** (*str*) – Email body
- **files** (*list*) – Optional list of files to attach

get_labels ()
Returns the available labels on the GMail account

2.37 platypush.plugins.google.maps

class platypush.plugins.google.maps.**GoogleMapsPlugin** (*api_key*, **args*, ***kwargs*)
Plugins that provides utilities to interact with Google Maps API services.

__init__ (*api_key*, **args*, ***kwargs*)

Parameters **api_key** (*str*) – Server-side API key to be used for the requests, get one at https://console.developers.google.com

get_address_from_latlng (*latitude*, *longitude*)
Get an address information given lat/long

Parameters

- **latitude** (*float*) – Latitude
- **longitude** (*float*) – Longitude

get_elevation_from_latlng (*latitude*, *longitude*)
Get the elevation in meters of a geo point given lat/long

Parameters

- **latitude** (*float*) – Latitude

- **`longitude`** (*`float`*) – Longitude

2.38 `platypush.plugins.google.pubsub`

class `platypush.plugins.google.pubsub.`**`GooglePubsubPlugin`**(*credentials_file: str = '/home/docs/.credentials/platypush/google/pubsub*
***kwargs*)

Send messages over a Google pub/sub instance. You'll need a Google Cloud active project and a set of credentials to use this plugin:

1. Create a project on the Google Cloud console if you don't have one already.
2. In the Google Cloud API console create a new service account key. Select "New Service Account", choose the role "Pub/Sub Editor" and leave the key type as JSON.
3. Download the JSON service credentials file. By default platypush will look for the credentials file under ~/.credentials/platypush/google/pubsub.json.

Requires:

- **google-cloud-pubsub** (`pip install google-cloud-pubsub`)

`__init__`(*credentials_file: str = '/home/docs/.credentials/platypush/google/pubsub.json'*, ***kwargs*)

Parameters **`credentials_file`** – Path to the JSON credentials file for Google pub/sub (default: ~/.credentials/platypush/google/pubsub.json)

`send_message`(*topic: str*, *msg*, ***kwargs*)

Sends a message to a topic

Parameters

- **`topic`** – Topic/channel where the message will be delivered. You can either specify the full topic name in the format `projects/<project_id>/topics/<topic_name>`, where `<project_id>` must be the ID of your Google Pub/Sub project, or just `<topic_name>` - in such case it's implied that you refer to the `topic_name` under the `project_id` of your service credentials.
- **`msg`** – Message to be sent. It can be a list, a dict, or a Message object
- **`kwargs`** – Extra arguments to be passed to .publish()

2.39 `platypush.plugins.google.translate`

class `platypush.plugins.google.translate.`**`GoogleTranslatePlugin`**(*target_language: str = 'en'*, *credentials_file: Optional[str] = None*, ***kwargs*)

Plugin to interact with the Google Translate API. You'll need a Google Cloud active project and a set of credentials to use this plugin:

1. Create a project on the Google Cloud console if you don't have one already.
2. In the menu navigate to the *Artificial Intelligence* section and select *Translations* and enable the API.

3. From the menu select *APIs & Services* and create a service account. You can leave role and permissions empty.
4. Create a new private JSON key for the service account and download it. By default platypush will look for the credentials file under `~/.credentials/platypush/google/translate.json`.

Requires:

- **google-cloud-translate** (`pip install google-cloud-translate`)

`__init__`(*target_language: str = 'en'*, *credentials_file: Optional[str] = None*, ***kwargs*)

Parameters

- **`target_language`** – Default target language (default: 'en').
- **`credentials_file`** – Google service account JSON credentials file. If none is specified then the plugin will search for the credentials file in the following order:
  1. `~/.credentials/platypush/google/translate.json`
  2. Context from the `GOOGLE_APPLICATION_CREDENTIALS` environment variable.

`translate`(*text: str*, *target_language: Optional[str] = None*, *source_language: Optional[str] = None*, *format: Optional[str] = None*) → platypush.message.response.translate.TranslateResponse

Translate a piece of text or HTML.

Parameters

- **`text`** – Input text.
- **`target_language`** – target_language override.
- **`source_language`** – source_language (default: auto-detect).
- **`format`** – Input format (available formats: `text`, `html`).

Returns *`platypush.message.response.translate.TranslateResponse`*.

2.40 `platypush.plugins.google.youtube`

class `platypush.plugins.google.youtube.`**`GoogleYoutubePlugin`**(**args*, ***kwargs*)

YouTube plugin

`__init__`(**args*, ***kwargs*)

Initialized the Google plugin with the required scopes.

Parameters **`scopes`** (*`list`*) – List of required scopes

`search`(*parts=None*, *query=''*, *types=None*, *max_results=25*, ***kwargs*)

Search for YouTube content.

Parameters

- **`parts`** (*`list[str] or str`*) – List of parts to get (default: snippet). See the Getting started - Part.
- **`query`** (*`str`*) – Query string (default: empty string)
- **`types`** (*`list[str] or str`*) – List of types to retrieve (default: video). See the Getting started - Resources.

- **max_results** (*int*) – Maximum number of items that will be returned (default: 25).
- **kwargs** – Any extra arguments that will be transparently passed to the YouTube API. See the Getting started - parameters.

Returns A list of YouTube resources. See the Getting started - Resource.

2.41 platypush.plugins.gpio

class platypush.plugins.gpio.**GpioPlugin**(*pins: Optional[Dict[str, int]] = None, mode: str = 'board', **kwargs*)

Plugin to handle raw read/write operation on the Raspberry Pi GPIO pins.

Requires:

- **RPi.GPIO** (*pip install RPi.GPIO*)

__init__(*pins: Optional[Dict[str, int]] = None, mode: str = 'board', **kwargs*)

Parameters

- **mode** – Specify 'board' if you want to use the board PIN numbers, 'bcm' for Broadcom PIN numbers (default: 'board')
- **pins** – Configuration for the GPIO PINs as a name -> pin_number map.

Example:

```
{
    "LED_1": 14,
    "LED_2": 15,
    "MOTOR": 16,
    "SENSOR": 17
}
```

cleanup()

Cleanup the state of the GPIO and resets PIN values.

read(*pin: Union[int, str], name: Optional[str] = None*) → Dict[str, Any]

Reads a value from a PIN.

Parameters

- **pin** – PIN number or configured name.
- **name** – Optional name for the read value (e.g. "temperature" or "humidity")

Response:

```
output = {
    "name": <pin number or pin/metric name>,
    "pin": <pin>,
    "value": <value>,
    "method": "read"
}
```

read_all()

Reads the values from all the configured PINs and returns them as a list. It will raise a RuntimeError if no PIN mappings were configured.

write(*pin: Union[int, str], value: Union[int, bool], name: Optional[str] = None*) → Dict[str, Any]
Write a byte value to a pin.

Parameters

- **pin** – PIN number or configured name
- **name** – Optional name for the written value (e.g. "temperature" or "humidity")
- **value** – Value to write

Response:

```
output = {
    "name": <pin or metric name>,
    "pin": <pin>,
    "value": <value>,
    "method": "write"
}
```

2.42 platypush.plugins.gpio.sensor

class platypush.plugins.gpio.sensor.**GpioSensorPlugin**(***kwargs*)

2.43 platypush.plugins.gpio.sensor.accelerometer

class platypush.plugins.gpio.sensor.accelerometer.**GpioSensorAccelerometerPlugin**(*g=4, precision=None, *args, **kwargs*)

Plugin to interact with an accelerometer sensor and get X,Y,Z position. Tested with Adafruit LIS3DH accelerometer (https://www.adafruit.com/product/2809) with Raspberry Pi over I2C connection.

Requires:

- `Adafruit_Python_GPIO` (`pip install Adafruit_Python_GPIO`)

__init__(*g=4, precision=None, *args, **kwargs*)
Only LIS3DH in I2C mode is currently supported: https://learn.adafruit.com/assets/59080.

Parameters

- **g** (*int*) – Accelerometer range as a multiple of G - can be 2G, 4G, 8G or 16G
- **precision** (*int*) – If set, the position values will be rounded to the specified number of decimal digits (default: no rounding)

get_measurement()
Extends `GpioSensorPlugin.get_measurement()`

Returns The sensor's current position as a dictionary with the three components (x,y,z) in degrees, each between -90 and 90

2.44 `platypush.plugins.gpio.sensor.bme280`

class `platypush.plugins.gpio.sensor.bme280.`**`GpioSensorBme280Plugin`**(*port=1*, ***kwargs*)

Plugin to interact with a BME280 environment sensor for temperature, humidity and pressure measurements over I2C interface

Requires:

- `pimoroni-bme280` (`pip install pimoroni-bme280`)

`__init__`(*port=1*, ***kwargs*)

Parameters **`port`** – I2C port. 0 = /dev/i2c-0 (port I2C0), 1 = /dev/i2c-1 (port I2C1)

`get_measurement`()

Returns dict. Example:

```
output = {
    "temperature": 21.0,    # Celsius
    "pressure": 101555.08,  # Pascals
    "humidity": 23.543,     # percentage
    "altitude": 15.703      # meters
}
```

2.45 `platypush.plugins.gpio.sensor.dht`

class `platypush.plugins.gpio.sensor.dht.`**`GpioSensorDhtPlugin`**(*sensor_type: str*, *pin: int*, *retries: int = 5*, *retry_seconds: int = 2*, ***kwargs*)

Plugin to interact with a DHT11/DHT22/AM2302 temperature/humidity sensor through GPIO.

Requires:

- `Adafruit_Python_DHT` (`pip install git+https://github.com/adafruit/Adafruit_Python_DHT.git`)

`__init__`(*sensor_type: str*, *pin: int*, *retries: int = 5*, *retry_seconds: int = 2*, ***kwargs*)

Parameters

- **`sensor_type`** – Type of sensor to be used (supported types: DHT11, DHT22, AM2302).
- **`pin`** – GPIO PIN where the sensor is connected.
- **`retries`** – Number of retries in case of failed read (default: 5).
- **`retry_seconds`** – Number of seconds to wait between retries (default: 2).

`get_measurement`() → Dict[str, float]

Get data from the sensor.

Returns

A mapping with the measured temperature and humidity. Example:

```
{
    "humidity": 30.0,
    "temperature": 25.5
}
```

read(*sensor_type: Optional[str] = None*, *pin: Optional[int] = None*, *retries: Optional[int] = None*, *retry_seconds: Optional[int] = None*, ***kwargs*) → Dict[str, float]

Read data from the sensor.

Parameters

- **sensor_type** – Default `sensor_type` override.
- **pin** – Default `pin` override.
- **retries** – Default `retries` override.
- **retry_seconds** – Default `retry_seconds` override.

Returns

A mapping with the measured temperature and humidity. Example:

```
{
    "humidity": 30.0,
    "temperature": 25.5
}
```

2.46 platypush.plugins.gpio.sensor.distance

class platypush.plugins.gpio.sensor.distance.**GpioSensorDistancePlugin**(*trigger_pin: int*, *echo_pin: int*, *measurement_interval: float = 0.15*, *timeout: float = 2.0*, *warmup_time: float = 2.0*, **args*, ***kwargs*)

You can use this plugin to interact with a distance sensor on your Raspberry Pi (tested with a HC-SR04 ultrasound sensor).

Requires:

- `RPi.GPIO` (`pip install RPi.GPIO`)

Triggers:

- *platypush.message.event.distance.DistanceSensorEvent* when a new distance measurement is available

__init__(*trigger_pin: int*, *echo_pin: int*, *measurement_interval: float = 0.15*, *timeout: float = 2.0*, *warmup_time: float = 2.0*, **args*, ***kwargs*)

Parameters

- **trigger_pin** – GPIO PIN where you connected your sensor trigger PIN (the one that triggers the sensor to perform a measurement).
- **echo_pin** – GPIO PIN where you connected your sensor echo PIN (the one that will listen for the signal to bounce back and therefore trigger the distance calculation).
- **measurement_interval** – When running in continuous mode (see *platypush.plugins.gpio.sensor.distance.GpioSensorDistancePlugin.start_measurement()*) this parameter indicates how long should be waited between two measurements (default: 0.15 seconds)
- **timeout** – The echo-wait will terminate and the plugin will return null if no echo has been received after this time (default: 1 second).
- **warmup_time** – Number of seconds that should be waited on plugin instantiation for the sensor to be ready (default: 2 seconds).

get_measurement()

Extends `GpioSensorPlugin.get_measurement()`

Returns Distance measurement as a scalar (in mm):

start_measurement(*duration: Optional[float] = None*)

Start the measurement thread. It will trigger *platypush.message.event.distance.DistanceSensorEvent* events when new measurements are available.

Parameters **duration** – If set, then the thread will run for the specified amount of seconds (default: None)

stop_measurement()

Stop the running measurement thread.

2.47 platypush.plugins.gpio.sensor.distance.vl53l1x

class platypush.plugins.gpio.sensor.distance.vl53l1x.**GpioSensorDistanceVl53L1XPlugin**(*i2c_bus*
i2c_add
***kwarg*

Plugin to interact with an VL53L1x laser ranger/distance sensor

Requires:

- `smbus2` (`pip install smbus2`)
- `vl53l1x` (`pip install vl53l1x`)

__init__(*i2c_bus=1*, *i2c_address=41*, ***kwargs*)

Parameters

- **i2c_bus** – I2C bus number (default: 1)
- **i2c_address** – I2C address (default: 0x29)

get_measurement(*short=True*, *medium=False*, *long=False*)

Parameters

- **short** – Enable short range measurement (default: True)
- **medium** – Enable medium range measurement (default: False)
- **long** – Enable long range measurement (default: False)

Returns dict. Example:

```
output = {
    "short": 83,      # Short range measurement in mm
    "medium": 103,    # Medium range measurement
    "long": 43,       # Long range measurement
}
```

2.48 platypush.plugins.gpio.sensor.envirophat

class platypush.plugins.gpio.sensor.envirophat.**GpioSensorEnvirophatPlugin**(***kwargs*)

Plugin to interact with a Pimoroni enviropHAT device. You can use an enviropHAT device to read e.g. temperature, pressure, altitude, accelerometer, magnetometer and luminosity data, plus control the status of its RGB LEDs.

Requires:

- `envirophat` (`pip install envirophat`)

get_measurement(*qnh=1020.0*)

Param qnh: Local value for atmospheric pressure adjusted to sea level (default: 1020)

Returns dict. Example:

```
output = {
    "temperature": 21.0,    # Celsius
    "pressure": 101555.08, # pascals
    "altitude": 10,         # meters
    "luminosity": 426,      # lumens

    # Measurements from the custom analog channels
    "analog": [0.513, 0.519, 0.531, 0.528],

    "accelerometer": {
        "x": -0.000915,
        "y": 0.0760,
        "z": 1.026733
    },
    "magnetometer": {
        "x": -2297,
        "y": 1226,
        "z": -7023
    },
}
```

2.49 `platypush.plugins.gpio.sensor.ltr559`

class `platypush.plugins.gpio.sensor.ltr559.`**`GpioSensorLtr559Plugin`**(***kwargs*)

Plugin to interact with an LTR559 light and proximity sensor

Requires:

- `ltr559` (`pip install ltr559`)

`__init__`(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

`get_measurement`()

Returns

dict. Example:

```
output = {
    "light": 109.3543,      # Lux
    "proximity": 103        # The higher the value, the nearest␣
↪the object, within a ~5cm range
}
```

2.50 `platypush.plugins.gpio.sensor.mcp3008`

class `platypush.plugins.gpio.sensor.mcp3008.`**`GpioSensorMcp3008Plugin`**(*CLK=None, MISO=None, MOSI=None, CS=None, spi_port=None, spi_device=None, channels=None, Vdd=3.3, *args, **kwargs*)

Plugin to read analog sensor values from an MCP3008 chipset. The MCP3008 chipset is a circuit that allows you to read measuremnts from multiple analog sources (e.g. sensors) and multiplex them to a digital device like a Raspberry Pi or a regular laptop. See https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/mcp3008 for more info.

Requires:

- `adafruit-mcp3008` (`pip install adafruit-mcp3008`)

`__init__`(*CLK=None, MISO=None, MOSI=None, CS=None, spi_port=None, spi_device=None, channels=None, Vdd=3.3, *args, **kwargs*)

The MCP3008 can be connected in two modes:

- **Hardware SPI mode: advised if you have enough GPIO pins available** (and slightly faster)
- **Software SPI mode: useful if you don't have all the required GPIO** PINs for hardware SPI available. Slightly slower, as the conversion is done via software, but still relatively performant.

See https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/mcp3008#wiring for info

Parameters

- **CLK** (*int*) – (software SPI mode) CLK GPIO PIN
- **MISO** (*int*) – (software SPI mode) MISO GPIO PIN
- **MOSI** (*int*) – (software SPI mode) MOSI GPIO PIN
- **CS** (*int*) – (software SPI mode) CS GPIO PIN
- **spi_port** (*int*) – (hardware SPI mode) SPI port
- **spi_device** (*str*) – (hardware SPI mode) SPI device name
- **channels** (*dict*) – name-value mapping between MCP3008 output PINs and sensor names. This mapping will be used when you get values through `get_measurement()`. Example:

```
channels = {
    "0": {
        "name": "temperature",
        "conv_function": 'round(x*100.0, 2)'  # T = Vout /␣
↪(10 [mV/C])
    },
    "1": {
        "name": "light",  # ALS-PT9
        "conv_function": 'round(x*1000.0, 6)'  # ALS-PT9 has␣
↪a 10 kOhm resistor
    }
}
```

  Note that you can also pass a conversion function as `conv_function` that will convert the output voltage to whichever human-readable value you wish. In the case above I connected a simple temperature sensor to the channel 0 and a simple ALS-PT9 light sensor to the channel 1, and passed the appropriate conversion functions to convert from voltage to, respectively, temperature in Celsius degrees and light intensity in lumen. Note that we reference the current voltage as `x` in `conv_function`.

- **Vdd** (*float*) – Input voltage provided to the circuit (default: 3.3V, Raspberry Pi default power source)

get_measurement()

Returns a measurement from the sensors connected to the MCP3008 device. If channels were passed to the configuration, the appropriate sensor names will be used and the voltage will be converted through the appropriate conversion function. Example:

```
output = {
    "temperature": 21.0,  # Celsius
    "humidity": 45.1     # %
}
```

Otherwise, the output dictionary will contain the channel numbers as key and the row voltage (between 0 and 255) will be returned. Example:

```
output = {
    "0": 145,
```

(continues on next page)

(continued from previous page)

```
    "1": 130
}
```

class platypush.plugins.gpio.sensor.mcp3008.**MCP3008Mode**
An enumeration.

2.51 platypush.plugins.gpio.sensor.motion.pwm3901

class platypush.plugins.gpio.sensor.motion.pwm3901.**GpioSensorMotionPwm3901Plugin**(*rotation=0, spi_slot='fron spi_port=0, **kwargs*)

Plugin to interact with an PWM3901 optical flow and motion sensor

Requires:

- pwm3901 (pip install pwm3901)

__init__(*rotation=0, spi_slot='front', spi_port=0, **kwargs*)

Parameters

- **rotation** (*int*) – Rotation angle for the captured optical flow. Possible options: 0, 90, 180, 270 (default: 0)
- **spi_slot** (*int*) – SPI slot where the sensor is connected if you're using a Breakout Garden interface. Possible options: 'front', 'back' (default: 'front')
- **spi_port** – SPI port (default: 0)

get_measurement()

Returns dict. Example:

```
output = {
    "motion_x": 3,    # Detected motion vector X-coord
    "motion_y": 4,    # Detected motion vector Y-coord
    "motion_mod": 5   # Detected motion vector module
    "motion_events_per_sec": 7  # Number of motion events detected in the␣
↪last second
}
```

class platypush.plugins.gpio.sensor.motion.pwm3901.**Rotation**
An enumeration.

class platypush.plugins.gpio.sensor.motion.pwm3901.**SPISlot**
An enumeration.

2.52 platypush.plugins.gpio.zeroborg

class platypush.plugins.gpio.zeroborg.**Direction**
An enumeration.

class platypush.plugins.gpio.zeroborg.**GpioZeroborgPlugin**(*directions: Dict[str, List[float]] = None, **kwargs*)

ZeroBorg plugin. It allows you to control a ZeroBorg (https://www.piborg.org/motor-control-1135/zeroborg) motor controller and infrared sensor circuitry for Raspberry Pi

Triggers:

- `platypush.message.event.zeroborg.ZeroborgDriveEvent` when motors direction changes
- `platypush.message.event.zeroborg.ZeroborgStopEvent` upon motors stop

__init__(*directions: Dict[str, List[float]] = None, **kwargs*)

Parameters **directions** – Configuration for the motor directions. A direction is basically a configuration of the power delivered to each motor to allow whichever object you're controlling (wheels, robotic arms etc.) to move in a certain direction. In my experience the ZeroBorg always needs a bit of calibration, depending on factory defaults and the mechanical properties of the load it controls.

Example configuration that I use to control a simple 4WD robot:

```
directions = {
    "up": [
        0.4821428571428572,    # Motor 1 power
        0.4821428571428572,    # Motor 2 power
        -0.6707142857142858,   # Motor 3 power
        -0.6707142857142858    # Motor 4 power
    ],
    "down": [
        -0.4821428571428572,
        -0.4821428571428572,
        0.825,
        0.825
    ],
    "left": [
        -0.1392857142857143,
        -0.1392857142857143,
        -1.0553571428571429,
        -1.0553571428571429
    ],
    "right": [
        1.0017857142857143,
        1.0017857142857143,
        0.6214285714285713,
        0.6214285714285713
    ]
}
```

drive(*direction*)

Drive the motors in a certain direction.

status() → dict

Get the current direction and motors power. Example response:

```
.. code-block:: json

{
    "status": "running",
```

(continues on next page)

(continued from previous page)

```
    "direction": "up",
    "motors": [1.0, 1.0, -1.0, -1.0]
}
```

stop()
Turns off the motors

2.53 platypush.plugins.graphite

class platypush.plugins.graphite.**GraphitePlugin**(*host: str = 'localhost'*, *port: int = 2003*, *timeout: int = 5*, ***kwargs*)

Plugin for sending data to a Graphite instance.

__init__(*host: str = 'localhost'*, *port: int = 2003*, *timeout: int = 5*, ***kwargs*)

Parameters

- **host** – Default Graphite host (default: 'localhost').
- **port** – Default Graphite port (default: 2003).
- **timeout** – Communication timeout in seconds (default: 5).

send(*metric: str*, *value*, *host: Optional[str] = None*, *port: Optional[int] = None*, *timeout: Optional[int] = None*, *tags: Optional[Dict[str, str]] = None*, *prefix: str = ''*, *protocol: str = 'tcp'*)
Send data to a Graphite instance.

Parameters

- **metric** – Metric name.
- **value** – Value to be sent.
- **host** – Graphite host (default: default configured `host`).
- **port** – Graphite port (default: default configured `port`).
- **tags** – Map of tags for the metric.
- **prefix** – Metric prefix name (default: empty string).
- **protocol** – Communication protocol - possible values: 'tcp', 'udp' (default: 'tcp').

2.54 platypush.plugins.homeseer

class platypush.plugins.homeseer.**HomeseerPlugin**(*host*, *username=None*, *password=None*, **args*, ***kwargs*)

This plugin allows you interact with an existing HomeSeer setup, query and control connected devices.

Requires:

- **pyhomeseer** (`pip install git+https://github.com/legrego/PyHomeSeer`)

__init__(*host*, *username=None*, *password=None*, **args*, ***kwargs*)

Parameters

- **host** (*str*) – IP or hostname of your HomeSeer hub
- **username** (*str*) – HomeSeer username

- **password** (*str*) – HomeSeer password

control (*ref*, *value=None*, *label=None*)
Control a HomeSeer connected device.

Parameters

- **ref** (*int*) – Device reference
- **value** (*int*) – If set, then control the device with this specific int value
- **label** (*str*) – If set, then control the device with this specific label (e.g. 'On' or 'Off')

query_devices (*ref=None*, *location=None*)
Get a list of devices connected to HomeSeer with their status

Parameters

- **ref** (*int*) – Device reference. If not set, all the devices will be queried
- **location** (*str*) – Device location. If not set, all the devices will be queried

2.55 platypush.plugins.http.request

class platypush.plugins.http.request.**HttpRequestPlugin**(***kwargs*)
Plugin for executing custom HTTP requests.

Requires:

- **requests** (`pip install requests`)

Some example usages:

```
# Execute a GET request on a JSON endpoint
{
    "type": "request",
    "action": "http.request.get",
    "args": {
        "url": "http://remote-host/api/v1/entity",
        "params": {
            "start": "2000-01-01"
        }
    }
}

# Execute an action on another Platypush host through HTTP interface
{
    "type": "request",
    "action": "http.request.post",
    "args": {
        "url": "http://remote-host:8008/execute",
        "json": {
            "type": "request",
            "target": "remote-host",
            "action": "music.mpd.play"
        }
    }
}
```

`__init__`(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

`delete`(*url*, ***kwargs*)
Perform a DELETE request

Parameters

- **`url`** (*`str`*) – Target URL
- **`kwargs`** (*`dict`*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

`download`(*url: str*, *path: str*, ***kwargs*)
Locally download the content of a remote URL.

Parameters

- **`url`** – URL to be downloaded.
- **`path`** – Path where the content will be downloaded on the local filesystem - must be a file name.

`get`(*url*, ***kwargs*)
Perform a GET request

Parameters

- **`url`** (*`str`*) – Target URL
- **`kwargs`** (*`dict`*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

`head`(*url*, ***kwargs*)
Perform an HTTP HEAD request

Parameters

- **`url`** (*`str`*) – Target URL
- **`kwargs`** (*`dict`*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

`options`(*url*, ***kwargs*)
Perform an HTTP OPTIONS request

Parameters

- **`url`** (*`str`*) – Target URL
- **`kwargs`** (*`dict`*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

`post`(*url*, ***kwargs*)
Perform a POST request

Parameters

- **`url`** (*`str`*) – Target URL

- **kwargs** (*dict*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

put (*url*, ***kwargs*)
Perform a PUT request

Parameters

- **url** (*str*) – Target URL
- **kwargs** (*dict*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

2.56 platypush.plugins.http.request.ota.booking

class `platypush.plugins.http.request.ota.booking.`**HttpRequestOtaBookingPlugin** (*hotel_id*, *token*, *timeout=5*, ***kwargs*)

Plugin to send requests to the Booking Hub API

__init__ (*hotel_id*, *token*, *timeout=5*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

2.57 platypush.plugins.http.request.rss

class `platypush.plugins.http.request.rss.`**HttpRequestRssPlugin** (***kwargs*)
Plugin to programmatically retrieve and parse an RSS feed URL.

Requires:

- **feedparser** (`pip install feedparser`)

get (*url*)
Perform a GET request

Parameters

- **url** (*str*) – Target URL
- **kwargs** (*dict*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

2.58 platypush.plugins.http.webpage

class `platypush.plugins.http.webpage.`**HttpWebpagePlugin** (***kwargs*)
Plugin to handle and parse/simplify web pages. It used to use the Mercury Reader web API, but now that the API is discontinued this plugin is basically a wrapper around the mercury-parser JavaScript library.

Requires:

- **requests** (`pip install requests`)
- **weasyprint** (`pip install weasyprint`), optional, for HTML->PDF conversion
- **node** and **npm** installed on your system (to use the mercury-parser interface)
- The mercury-parser library installed (`npm install @postlight/mercury-parser`)

`simplify`(*url*, *type='html'*, *html=None*, *outfile=None*)

Parse the content of a web page removing any extra elements using Mercury

Parameters

- **`url`** – URL to parse.
- **`type`** – Input type. Supported types: html, markdown, text (default: html).
- **`html`** – Set this parameter if you want to parse some HTML content already fetched. Note that URL is still required by Mercury to properly style the output, but it won't be used to actually fetch the content.
- **`outfile`** – If set then the output will be written to the specified file (supported formats: pdf, html, plain (default)). The plugin will guess the format from the extension

Returns dict

Example if outfile is not specified:

```
{
    "url": <url>,
    "title": <page title>,
    "content": <page parsed content>

}
```

Example if outfile is specified:

```
{
    "url": <url>,
    "title": <page title>,
    "outfile": <output file absolute path>

}
```

2.59 `platypush.plugins.ifttt`

class `platypush.plugins.ifttt.`**`IftttPlugin`**(*ifttt_key*, ***kwargs*)

This plugin allows you to interact with the IFTTT maker API <https://ifttt.com/maker_webhooks> to programmatically trigger your own IFTTT hooks from Platypush - e.g. send a tweet or a Facebook post, create a Todoist item or a Trello task, trigger events on your mobile device, or run any action not natively supported by Platypush but available on your IFTTT configuration.

Requires:

- **requests** (`pip install requests`)

An example:

```
# Trigger an IFTTT event named "at_home"
{
    "type": "request",
    "action": "ifttt.trigger_event",
    "args": {
        "event_name": "at_home"
    }
}
```

`__init__`(*ifttt_key*, ***kwargs*)

Parameters **`ifttt_key`** (`str`) – Your IFTTT Maker API key. Log in to IFTTT and get your key from here. Once you've got your key, you can start creating IFTTT rules using the Webhooks channel.

`trigger_event`(*event_name*, *values=None*)

Send an event to your IFTTT account

Parameters

- **`event_name`** (`str`) – Name of the event
- **`values`** (`list`) – Optional list of values to be passed to the event. By convention IFTTT names the values as `value1,value2,...`.

2.60 `platypush.plugins.inputs`

class `platypush.plugins.inputs.`**`InputsPlugin`**(***kwargs*)

This plugin emulates user input on a keyboard/mouse. It requires the a graphical server (X server or Mac/Win interface) to be running - it won't work in console mode.

Requires:

- **pyuserinput** (`pip install pyuserinput`)

`get_screen_size`() → List[int]

Get the size of the screen in pixels.

`mouse_click`(*x: int*, *y: int*, *btn: int*, *repeat: int = 1*)

Mouse click. :param x: x screen position :param y: y screen position :param btn: Button number (1 for left, 2 for right, 3 for middle) :param repeat: Number of clicks (default: 1)

`press_key`(*key: str*)

Emulate the pressure of a key. :param key: Key to be pressed

`press_keys`(*keys: List[str]*)

Emulate the pressure of multiple keys. :param keys: List of keys to be pressed.

`release_key`(*key: str*)

Release a pressed key. :param key: Key to be released

`tap_key`(*key: str*, *repeat: int = 1*, *interval: float = 0*)

Emulate a key tap. :param key: Key to be pressed :param repeat: Number of iterations (default: 1) :param interval: Repeat interval in seconds (default: 0)

`type_string`(*string: str*, *interval: float = 0*)

Type a string. :param string: String to be typed :param interval: Interval between key strokes in seconds (default: 0)

2.61 `platypush.plugins.inspect`

class platypush.plugins.inspect.**InspectPlugin**(***kwargs*)

This plugin can be used to inspect platypush plugins and backends

Requires:

- **docutils** (`pip install docutils`) - optional, for HTML doc generation

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

get_all_backends(*html_doc: bool = None*)

Parameters html_doc – If True then the docstring will be parsed into HTML (default: False)

get_all_events(*html_doc: bool = None*)

Parameters html_doc – If True then the docstring will be parsed into HTML (default: False)

get_all_plugins(*html_doc: bool = None*)

Parameters html_doc – If True then the docstring will be parsed into HTML (default: False)

get_all_responses(*html_doc: bool = None*)

Parameters html_doc – If True then the docstring will be parsed into HTML (default: False)

get_config(*entry: Optional[str] = None*) → dict

Return the configuration of the application or of a section.

Parameters entry – [Optional] configuration entry name to retrieve (e.g. `workdir` or `backend.http`).

Returns The requested configuration object.

get_procedures() → dict

Get the list of procedures installed on the device.

class platypush.plugins.inspect.**ProcedureEncoder**(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

default(*o*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
```

(continues on next page)

(continued from previous page)

```
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

2.62 `platypush.plugins.kafka`

class `platypush.plugins.kafka.`**`KafkaPlugin`**(*server=None*, ***kwargs*)

Plugin to send messages to an Apache Kafka instance (https://kafka.apache.org/)

Triggers:

- *`platypush.message.event.kafka.KafkaMessageEvent`* when a new message is received on the consumer topic.

Requires:

- **kafka** (`pip install kafka-python`)

`__init__`(*server=None*, ***kwargs*)

Parameters **`server`** (*`str`*) – Default Kafka server name or address + port (format: `host:port`) to dispatch the messages to. If None (default), then it has to be specified upon message sending.

`send_message`(*msg*, *topic*, *server=None*, ***kwargs*)

Parameters

- **`msg`** – Message to send - as a string, bytes stream, JSON, Platypush message, dictionary, or anything that implements `__str__`
- **`server`** (*`str`*) – Kafka server name or address + port (format: `host:port`). If None, then the default server will be used

2.63 `platypush.plugins.lastfm`

class `platypush.plugins.lastfm.`**`LastfmPlugin`**(*api_key*, *api_secret*, *username*, *password*)

Plugin to interact with your Last.FM (https://last.fm) account, update your current track and your scrobbles.

Requires:

- **pylast** (`pip install pylast`)

`__init__`(*api_key*, *api_secret*, *username*, *password*)

Parameters

- **`api_key`** (*`str`*) – Last.FM API key, see https://www.last.fm/api
- **`api_secret`** – Last.FM API secret, see https://www.last.fm/api
- **`username`** – Last.FM username
- **`password`** – Last.FM password, used to sign the requests

`scrobble`(*artist*, *title*, *album=None*, ***kwargs*)

Scrobble a track to Last.FM

Parameters

- **artist** (*str*) – Artist
- **title** (*str*) – Title
- **album** (*str*) – Album (optional)

update_now_playing(*artist*, *title*, *album=None*, ***kwargs*)
Update the currently playing track

Parameters

- **artist** (*str*) – Artist
- **title** (*str*) – Title
- **album** (*str*) – Album (optional)

2.64 platypush.plugins.lcd

class platypush.plugins.lcd.**LcdPlugin**(***kwargs*)
Abstract class for plugins to communicate with LCD displays.

Requires:

- **RPLCD** (`pip install RPLCD`)
- **RPi.GPIO** (`pip install RPi.GPIO`)

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

clear()
Clear the LCD display.

close(*clear: bool = False*)
Close the handler to the LCD display and release the GPIO resources.

Parameters clear – Clear the display as well on close (default: False).

command(*value: int*)
Send a raw command to the LCD.

Parameters value – Command to be sent.

cr()
Write a carriage return (`\r`) character to the LCD.

create_char(*location: int, bitmap: List[int]*)
Create a new character. The HD44780 supports up to 8 custom characters (location 0-7).

Parameters

- **location** – The place in memory where the character is stored. Values need to be integers between 0 and 7.
- **bitmap** – The bitmap containing the character. This should be a list of 8 numbers, each representing a 5 pixel row.

Example for the smiley character:

```
[
    0,    # 0b00000
    10,   # 0b01010
    10,   # 0b01010
    0,    # 0b00000
    17,   # 0b10001
    17,   # 0b10001
    14,   # 0b01110
    0     # 0b00000
]
```

`crlf`()
Write a carriage return + line feed (`\r\n`) sequence to the LCD.

`disable_backlight`()
Disable the display backlight.

`disable_display`()
Turn off the display.

`enable_backlight`()
Enable the display backlight.

`enable_display`()
Turn on the display.

`home`()
Set cursor to initial position and reset any shifting.

`lf`()
Write a line feed (`\n`) character to the LCD.

`set_cursor_pos`(*position: List[int]*)
Change the position of the cursor on the display.

Parameters **`position`** – New cursor position, as a list of two elements.

`set_text_align`(*mode: str*)
Change the text align mode.

Parameters **`mode`** – Supported values: `left`, `right`.

`shift_display`(*amount: int*)
Set cursor to initial position and reset any shifting.

`toggle_backlight`()
Toggle the display backlight on/off.

`toggle_display`()
Toggle the display state.

`write`(*value: int*)
Write a raw byte to the LCD.

Parameters **`value`** – Byte to be sent.

`write_string`(*value: str*, *position: Optional[List[int]] = None*)
Write a string to the display.

Parameters

- **`value`** – String to be displayed.
- **`position`** – String position on the display as a 2-int list.

2.65 `platypush.plugins.lcd.gpio`

class platypush.plugins.lcd.gpio.**LcdGpioPlugin**(*pin_rs: int, pin_e: int, pins_data: List[int], pin_rw: Optional[int] = None, pin_mode: str = 'BOARD', pin_backlight: Optional[int] = None, cols: int = 16, rows: int = 2, backlight_enabled: bool = True, backlight_mode: str = 'active_low', dotsize: int = 8, charmap: str = 'A02', auto_linebreaks: bool = True, compat_mode: bool = False, **kwargs*)

Plugin to write to an LCD display connected via GPIO.

Requires:

- **RPLCD** (`pip install RPLCD`)
- **RPi.GPIO** (`pip install RPi.GPIO`)

__init__(*pin_rs: int, pin_e: int, pins_data: List[int], pin_rw: Optional[int] = None, pin_mode: str = 'BOARD', pin_backlight: Optional[int] = None, cols: int = 16, rows: int = 2, backlight_enabled: bool = True, backlight_mode: str = 'active_low', dotsize: int = 8, charmap: str = 'A02', auto_linebreaks: bool = True, compat_mode: bool = False, **kwargs*)

Parameters

- **pin_rs** – Pin for register select (RS).
- **pin_e** – Pin to start data read or write (E).
- **pins_data** – List of data bus pins in 8 bit mode (DB0-DB7) or in 4 bit mode (DB4-DB7) in ascending order.
- **pin_mode** – Which scheme to use for numbering of the GPIO pins, either `BOARD` or `BCM`. Default: `BOARD`.
- **pin_rw** – Pin for selecting read or write mode (R/W). Default: `None`, read only mode.
- **pin_backlight** – Pin for controlling backlight on/off. Set this to `None` for no backlight control. Default: `None`.
- **cols** – Number of columns per row (usually 16 or 20). Default: `16`.
- **rows** – Number of display rows (usually 1, 2 or 4). Default: `2`.
- **backlight_enabled** – Whether the backlight is enabled initially. Default: `True`. Has no effect if pin_backlight is `None`
- **backlight_mode** – Set this to either `active_high` or `active_low` to configure the operating control for the backlight. Has no effect if pin_backlight is `None`
- **dotsize** – Some 1 line displays allow a font height of 10px. Allowed: `8` or `10`. Default: `8`.
- **charmap** – The character map used. Depends on your LCD. This must be either `A00` or `A02` or `ST0B`. Default: `A02`.
- **auto_linebreaks** – Whether or not to automatically insert line breaks. Default: `True`.

- **`compat_mode`** – Whether to run additional checks to support older LCDs that may not run at the reference clock (or keep up with it). Default: `False`.

2.66 `platypush.plugins.lcd.i2c`

class `platypush.plugins.lcd.i2c.`**`LcdI2cPlugin`**(*i2c_expander: str*, *address: int*, *expander_params: Optional[dict] = None*, *port: int = 1*, *cols: int = 16*, *rows: int = 2*, *backlight_enabled: bool = True*, *dotsize: int = 8*, *charmap: str = 'A02'*, *auto_linebreaks: bool = True*, ***kwargs*)

Plugin to write to an LCD display connected via I2C. Adafruit I2C/SPI LCD Backback is supported.

Warning: You might need a level shifter (that supports i2c) between the SCL/SDA connections on the MCP chip / backpack and the Raspberry Pi. Or you might damage the Pi and possibly any other 3.3V i2c devices connected on the i2c bus. Or cause reliability issues. The SCL/SDA are rated 0.7*VDD on the MCP23008, so it needs 3.5V on the SCL/SDA when 5V is applied to drive the LCD. The MCP23008 and MCP23017 needs to be connected exactly the same way as the backpack. For complete schematics see the adafruit page at: https://learn.adafruit.com/i2c-spi-lcd-backpack/ 4-bit operation. I2C only supported.

Pin mapping:

```
7  | 6  | 5  | 4  | 3  | 2 | 1  | 0
BL | D7 | D6 | D5 | D4 | E | RS | -
```

Requires:

- **RPLCD** (`pip install RPLCD`)
- **RPi.GPIO** (`pip install RPi.GPIO`)

`__init__`(*i2c_expander: str*, *address: int*, *expander_params: Optional[dict] = None*, *port: int = 1*, *cols: int = 16*, *rows: int = 2*, *backlight_enabled: bool = True*, *dotsize: int = 8*, *charmap: str = 'A02'*, *auto_linebreaks: bool = True*, ***kwargs*)

Parameters

- **`i2c_expander`** – Set your I^{2}C chip type. Supported: “PCF8574”, “MCP23008”, “MCP23017”.
- **`address`** – The I2C address of your LCD.
- **`expander_params`** – Parameters for expanders, in a dictionary. Only needed for MCP23017 gpio_bank - This must be either `A` or `B`. If you have a HAT, A is usually marked 1 and B is 2. Example: `expander_params={'gpio_bank': 'A'}`
- **`port`** – The I2C port number. Default: `1`.
- **`cols`** – Number of columns per row (usually 16 or 20). Default: `16`.
- **`rows`** – Number of display rows (usually 1, 2 or 4). Default: `2`.
- **`backlight_enabled`** – Whether the backlight is enabled initially. Default: `True`. Has no effect if pin_backlight is `None`
- **`dotsize`** – Some 1 line displays allow a font height of 10px. Allowed: `8` or `10`. Default: `8`.
- **`charmap`** – The character map used. Depends on your LCD. This must be either `A00` or `A02` or `ST0B`. Default: `A02`.

- **`auto_linebreaks`** – Whether or not to automatically insert line breaks. Default: `True`.

2.67 `platypush.plugins.light`

class `platypush.plugins.light.`**`LightPlugin`**(***kwargs*)

Abstract plugin to interface your logic with lights/bulbs.

`off`()

Turn the light off

`on`()

Turn the light on

`toggle`()

Toggle the light status (on/off)

2.68 `platypush.plugins.light.hue`

class `platypush.plugins.light.hue.`**`LightHuePlugin`**(*bridge*, *lights=None*, *groups=None*)

Philips Hue lights plugin.

Requires:

- **phue** (`pip install phue`)

Triggers:

- *`platypush.message.event.light.LightAnimationStartedEvent`* when an animation is started.
- *`platypush.message.event.light.LightAnimationStoppedEvent`* when an animation is stopped.

class Animation

An enumeration.

`__init__`(*bridge*, *lights=None*, *groups=None*)

Parameters

- **`bridge`** (*`str`*) – Bridge address or hostname
- **`lights`** (*`list[str]`*) – Default lights to be controlled (default: all)
- **`Default groups to be controlled (default`** (*`groups`*) – all)

`animate`(*animation*, *duration=None*, *hue_range=None*, *sat_range=None*, *bri_range=None*, *lights=None*, *groups=None*, *hue_step=1000*, *sat_step=2*, *bri_step=1*, *transition_seconds=1.0*)

Run a lights animation.

Parameters

- **`animation`** (*`str`*) – Animation name. Supported types: **color_transition** and **blink**
- **`duration`** (*`float`*) – Animation duration in seconds (default: None, i.e. continue until stop)

- **`hue_range`** (*`list[int]`*) – If you selected a `color_transition`, this will specify the hue range of your color `color_transition`. Default: [0, 65535]
- **`sat_range`** (*`list[int]`*) – If you selected a color `color_transition`, this will specify the saturation range of your color `color_transition`. Default: [0, 255]
- **`bri_range`** – If you selected a color `color_transition`, this will specify the brightness range of your color `color_transition`. Default: [254, 255] :type bri_range: list[int]
- **`lights`** – Lights to control (names, IDs or light objects). Default: plugin default lights
- **`groups`** – Groups to control (names, IDs or group objects). Default: plugin default groups
- **`hue_step`** – If you selected a color `color_transition`, this will specify by how much the color hue will change between iterations. Default: 1000 :type hue_step: int
- **`sat_step`** – If you selected a color `color_transition`, this will specify by how much the saturation will change between iterations. Default: 2 :type sat_step: int
- **`bri_step`** – If you selected a color `color_transition`, this will specify by how much the brightness will change between iterations. Default: 1 :type bri_step: int
- **`transition_seconds`** (*`float`*) – Time between two transitions or blinks in seconds. Default: 1.0

`bri`(*value*, *lights=None*, *groups=None*, ***kwargs*)
Set lights/groups brightness.

Parameters

- **`lights`** – Lights to control (names or light objects). Default: plugin default lights
- **`groups`** – Groups to control (names or group objects). Default: plugin default groups
- **`value`** – Brightness value (range: 0-255)

`connect`()
Connect to the configured Hue bridge. If the device hasn't been paired yet, uncomment the `.connect()` and `.get_api()` lines and retry after clicking the pairing button on your bridge.

`ct`(*value*, *lights=None*, *groups=None*, ***kwargs*)
Set lights/groups color temperature.

Parameters

- **`value`** (*`int`*) – Temperature value (range: 0-255)
- **`lights`** – List of lights.
- **`groups`** – List of groups.

`delta_bri`(*delta*, *lights=None*, *groups=None*, ***kwargs*)
Change lights/groups brightness by a delta [-100, 100] compared to the current state.

Parameters

- **`lights`** – Lights to control (names or light objects). Default: plugin default lights
- **`groups`** – Groups to control (names or group objects). Default: plugin default groups
- **`delta`** – Brightness delta value (range: -100, 100)

delta_hue (*delta*, *lights=None*, *groups=None*, ***kwargs*)
Change lights/groups hue by a delta [-100, 100] compared to the current state.

Parameters

- **lights** – Lights to control (names or light objects). Default: plugin default lights
- **groups** – Groups to control (names or group objects). Default: plugin default groups
- **delta** – Hue delta value (range: -100, 100)

delta_sat (*delta*, *lights=None*, *groups=None*, ***kwargs*)
Change lights/groups saturation by a delta [-100, 100] compared to the current state.

Parameters

- **lights** – Lights to control (names or light objects). Default: plugin default lights
- **groups** – Groups to control (names or group objects). Default: plugin default groups
- **delta** – Saturation delta value (range: -100, 100)

get_animations ()
Get the list of running light animations.

Returns dict.

Structure:

```
{
    "groups": {
        "id_1": {
            "type": "color_transition",
            "hue_range": [0,65535],
            "sat_range": [0,255],
            "bri_range": [0,255],
            "hue_step": 10,
            "sat_step": 10,
            "bri_step": 2,
            "transition_seconds": 2

        }

    },

    "lights": {
        "id_1": {}

    }

}
```

get_groups ()
Get the list of configured light groups.

Returns List of configured light groups as id->dict.

Example:

```
{
    "1": {
        "name": "Living Room",
        "lights": [
```

(continues on next page)

(continued from previous page)

```
                "16",
                "13",
                "12",
                "11",
                "10",
                "9",
                "1",
                "3"
            ],

            "type": "Room",
            "state": {
                "all_on": true,
                "any_on": true
            },

            "class": "Living room",
            "action": {
                "on": true,
                "bri": 241,
                "hue": 37947,
                "sat": 221,
                "effect": "none",
                "xy": [
                    0.2844,
                    0.2609
                ],

                "ct": 153,
                "alert": "none",
                "colormode": "hs"
            }
        }
}
```

get_lights()

Get the configured lights.

Returns List of available lights as id->dict.

Example:

```
{
    "1": {
        "state": {
            "on": true,
            "bri": 254,
            "hue": 1532,
            "sat": 215,
            "effect": "none",
            "xy": [
                0.6163,
                0.3403
            ],

            "ct": 153,
            "alert": "none",
```

(continues on next page)

(continued from previous page)

```
                "colormode": "hs",
                "reachable": true
            },

            "type": "Extended color light",
            "name": "Lightbulb 1",
            "modelid": "LCT001",
            "manufacturername": "Philips",
            "uniqueid": "00:11:22:33:44:55:66:77-88",
            "swversion": "5.105.0.21169"
        }
}
```

get_scenes()
Get the available scenes on the devices.

Returns The scenes configured on the bridge.

Example output:

```
{
    "scene-id-1": {
        "name": "Scene 1",
        "lights": [
            "1",
            "3"
        ],

        "owner": "owner-id",
        "recycle": true,
        "locked": false,
        "appdata": {},
        "picture": "",
        "lastupdated": "2018-06-01T00:00:00",
        "version": 1
    }
}
```

hue(*value*, *lights=None*, *groups=None*, ***kwargs*)
Set lights/groups color hue.

Parameters

- **lights** – Lights to control (names or light objects). Default: plugin default lights
- **groups** – Groups to control (names or group objects). Default: plugin default groups
- **value** – Hue value (range: 0-65535)

is_animation_running()

Returns True if there is an animation running, false otherwise.

off(*lights=None*, *groups=None*, ***kwargs*)
Turn lights/groups off.

Parameters

- **lights** – Lights to turn off (names or light objects). Default: plugin default lights

- **groups** – Groups to turn off (names or group objects). Default: plugin default groups

on (*lights=None*, *groups=None*, ***kwargs*)
Turn lights/groups on.

Parameters

- **lights** – Lights to turn on (names or light objects). Default: plugin default lights
- **groups** – Groups to turn on (names or group objects). Default: plugin default groups

sat (*value*, *lights=None*, *groups=None*, ***kwargs*)
Set lights/groups saturation.

Parameters

- **lights** – Lights to control (names or light objects). Default: plugin default lights
- **groups** – Groups to control (names or group objects). Default: plugin default groups
- **value** – Saturation value (range: 0-255)

scene (*name*, *lights=None*, *groups=None*, ***kwargs*)
Set a scene by name.

Parameters

- **lights** – Lights to control (names or light objects). Default: plugin default lights
- **groups** – Groups to control (names or group objects). Default: plugin default groups
- **name** – Name of the scene

set_group (*group*, ***kwargs*)
Set a group (or groups) property.

Parameters

- **group** – Group or groups to set. Can be a string representing the group name, a group object, a list of strings, or a list of group objects.
- **kwargs** – key-value list of parameters to set.

Example call:

```
{
    "type": "request",
    "target": "hostname",
    "action": "light.hue.set_group",
    "args": {
        "light": "Living Room",
        "sat": 255
    }
}
```

set_light (*light*, ***kwargs*)
Set a light (or lights) property.

Parameters

- **light** – Light or lights to set. Can be a string representing the light name, a light object, a list of string, or a list of light objects.
- **kwargs** – key-value list of parameters to set.

Example call:

```
{
    "type": "request",
    "target": "hostname",
    "action": "light.hue.set_light",
    "args": {
        "light": "Bulb 1",
        "sat": 255
    }
}
```

stop_animation()

Stop a running animation.

switches

Returns

Implements *platypush.plugins.switch.SwitchPlugin.switches()* and returns the status of the configured lights. Example:

```
[
    {
        "id": "3",
        "name": "Lightbulb 1",
        "on": true,
        "bri": 254,
        "hue": 1532,
        "sat": 215,
        "effect": "none",
        "xy": [
            0.6163,
            0.3403
        ],
        "ct": 153,
        "alert": "none",
        "colormode": "hs",
        "reachable": true
        "type": "Extended color light",
        "modelid": "LCT001",
        "manufacturername": "Philips",
        "uniqueid": "00:11:22:33:44:55:66:77-88",
        "swversion": "5.105.0.21169"
    }
]
```

toggle(*lights=None*, *groups=None*, ***kwargs*)

Toggle lights/groups on/off.

Parameters

- **lights** – Lights to turn off (names or light objects). Default: plugin default lights
- **groups** – Groups to turn off (names or group objects). Default: plugin default groups

xy(*value*, *lights=None*, *groups=None*, ***kwargs*)

Set lights/groups XY colors.

Parameters

- **value** (*list[float] containing the two values*) – xY value
- **lights** – List of lights.
- **groups** – List of groups.

2.69 platypush.plugins.linode

class platypush.plugins.linode.**LinodePlugin**(*token: str*, ***kwargs*)

This plugin can interact with a Linode account and manage node and volumes.

To get your token:

- Login to <https://cloud.linode.com/>.
- Go to My Profile -> API Tokens -> Add a Personal Access Token.
- Select the scopes that you want to provide to your new token.

Requires:

- **linode_api4** (`pip install linode_api4`)

__init__(*token: str*, ***kwargs*)

Parameters token – Your Linode token.

boot(*instance: str*, *token: Optional[str] = None*) → None

Boot an instance.

Parameters

- **instance** – Label of the instance to be booted.
- **token** – Default access token override.

get_measurement(**args*, ***kwargs*)

Implemented by the subclasses.

Returns

Either a raw scalar:

`output = 273.16`

or a name-value dictionary with the values that have been read:

```
output = {
    "temperature": 21.5,
    "humidity": 41.0
}
```

or a list of values:

```
[
    0.01,
    0.34,
    0.53,
    ...
]
```

reboot(*instance: str*, *token: Optional[str] = None*) → None

Reboot an instance.

Parameters

- **instance** – Label of the instance to be rebooted.
- **token** – Default access token override.

shutdown(*instance: str*, *token: Optional[str] = None*) → None
Shutdown an instance.

Parameters

- **instance** – Label of the instance to be shut down.
- **token** – Default access token override.

status(*token: Optional[str] = None*, *instance: Optional[str] = None*) → Union[platypush.message.response.linode.LinodeInstanceResponse, platypush.message.response.linode.LinodeInstancesResponse]
Get the full status and info of the instances associated to a selected account.

Parameters

- **token** – Override the default access token if you want to query another account.
- **instance** – Select only one node by label.

Returns `platypush.message.response.linode.LinodeInstanceResponse` if `label` is specified, `platypush.message.response.linode.LinodeInstancesResponse` otherwise.

2.70 platypush.plugins.logger

class platypush.plugins.logger.**LoggerPlugin**(***kwargs*)
Plugin to log traces on the standard Platypush logger

debug(*msg*, **args*, ***kwargs*)
logger.debug wrapper

error(*msg*, **args*, ***kwargs*)
logger.error wrapper

info(*msg*, **args*, ***kwargs*)
logger.info wrapper

trace(*msg*, **args*, ***kwargs*)
logger.trace wrapper

warning(*msg*, **args*, ***kwargs*)
logger.warning wrapper

2.71 platypush.plugins.luma.oled

class platypush.plugins.luma.oled.**DeviceInterface**
An enumeration.

class platypush.plugins.luma.oled.**DeviceRotation**
An enumeration.

class platypush.plugins.luma.oled.**DeviceSlot**
An enumeration.

class platypush.plugins.luma.oled.**LumaOledPlugin**(*interface: str*, *device: str*, *port: int = 0*, *slot: int = 0*, *width: int = 128*, *height: int = 64*, *rotate: int = 0*, *gpio_DC: int = 24*, *gpio_RST: int = 25*, *bus_speed_hz: int = 8000000*, *address: int = 60*, *cs_high: bool = False*, *transfer_size: int = 4096*, *spi_mode: Optional[int] = None*, *font: Optional[str] = None*, *font_size: int = 10*, ***kwargs*)

Plugin to interact with small OLED-based RaspberryPi displays through the luma.oled driver.

Requires:

- **luma.oled** (`pip install git+https://github.com/rm-hull/luma.oled`)

__init__(*interface: str*, *device: str*, *port: int = 0*, *slot: int = 0*, *width: int = 128*, *height: int = 64*, *rotate: int = 0*, *gpio_DC: int = 24*, *gpio_RST: int = 25*, *bus_speed_hz: int = 8000000*, *address: int = 60*, *cs_high: bool = False*, *transfer_size: int = 4096*, *spi_mode: Optional[int] = None*, *font: Optional[str] = None*, *font_size: int = 10*, ***kwargs*)

Parameters

- **interface** – Serial interface the display is connected to (`spi` or `i2c`).
- **device** – Display chipset type (supported: ssd1306 ssd1309, ssd1322, ssd1325, ssd1327, ssd1331, ssd1351, ssd1362, sh1106). :param port: Device port (usually 0 or 1).
- **slot** – Device slot (0 for back, 1 for front).
- **width** – Display width.
- **height** – Display height.
- **rotate** – Display rotation (0 for no rotation, 1 for 90 degrees, 2 for 180 degrees, 3 for 270 degrees).
- **gpio_DC** – [SPI only] GPIO PIN used for data (default: 24).
- **gpio_RST** – [SPI only] GPIO PIN used for RST (default: 25).
- **bus_speed_hz** – [SPI only] Bus speed in Hz (default: 8 MHz).
- **address** – [I2C only] Device address (default: 0x3c).
- **cs_high** – [SPI only] Set to True if the SPI chip select is high.
- **transfer_size** – [SPI only] Maximum amount of bytes to transfer in one go (default: 4096).
- **spi_mode** – [SPI only] SPI mode as two bit pattern of clock polarity and phase [CPOL|CPHA], 0-3 (default:None).
- **font** – Path to a default TTF font used to display the text.
- **font_size** – Font size - it only applies if `font` is set.

arc(*start: int, end: int, xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, outline: Optional[str] = None, width: int = 1, clear: bool = False*)

Draw an arc on the canvas.

Parameters

- **start** – Starting angle, in degrees (measured from 3 o' clock and increasing clockwise).
- **end** – Ending angle, in degrees (measured from 3 o' clock and increasing clockwise).
- **xy** – Two points defining the bounding box, either as [(x0, y0), (x1, y1)] or [x0, y0, x1, y1]. Default: bounding box of the device.
- **fill** – Fill color - can be `black` or `white`.
- **outline** – Outline color - can be `black` or `white`.
- **width** – Figure width in pixels (default: 1).
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

chord(*start: int, end: int, xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, outline: Optional[str] = None, width: int = 1, clear: bool = False*)

Same as `arc`, but it connects the end points with a straight line.

Parameters

- **start** – Starting angle, in degrees (measured from 3 o' clock and increasing clockwise).
- **end** – Ending angle, in degrees (measured from 3 o' clock and increasing clockwise).
- **xy** – Two points defining the bounding box, either as [(x0, y0), (x1, y1)] or [x0, y0, x1, y1]. Default: bounding box of the device.
- **fill** – Fill color - can be `black` or `white`.
- **outline** – Outline color - can be `black` or `white`.
- **width** – Figure width in pixels (default: 1).
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

clear()

clear the display canvas.

ellipse(*xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, outline: Optional[str] = None, width: int = 1, clear: bool = False*)

Draw an ellipse on the canvas.

Parameters

- **xy** – Two points defining the bounding box, either as [(x0, y0), (x1, y1)] or [x0, y0, x1, y1]. Default: bounding box of the device.
- **fill** – Fill color - can be `black` or `white`.
- **outline** – Outline color - can be `black` or `white`.
- **width** – Figure width in pixels (default: 1).
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

image(*image: str*)

Draws an image to the canvas (this will clear the existing canvas).

Parameters **image** – Image path.

line (*xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, outline: Optional[str] = None, width: int = 1, curve: bool = False, clear: bool = False*)
Draw a line on the canvas.

Parameters

- **xy** – Sequence of either 2-tuples like [(x, y), (x, y), . . .] or numeric values like [x, y, x, y, . . .].
- **fill** – Fill color - can be `black` or `white`.
- **outline** – Outline color - can be `black` or `white`.
- **width** – Figure width in pixels (default: 1).
- **curve** – Set to True for rounded edges (default: False).
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

pieslice (*start: int, end: int, xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, outline: Optional[str] = None, width: int = 1, clear: bool = False*)
Same as `arc`, but it also draws straight lines between the end points and the center of the bounding box.

Parameters

- **start** – Starting angle, in degrees (measured from 3 o' clock and increasing clockwise).
- **end** – Ending angle, in degrees (measured from 3 o' clock and increasing clockwise).
- **xy** – Two points defining the bounding box, either as [(x0, y0), (x1, y1)] or [x0, y0, x1, y1]. Default: bounding box of the device.
- **fill** – Fill color - can be `black` or `white`.
- **outline** – Outline color - can be `black` or `white`.
- **width** – Figure width in pixels (default: 1).
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

point (*xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, clear: bool = False*)
Draw one or more points on the canvas.

Parameters

- **xy** – Sequence of either 2-tuples like [(x, y), (x, y), . . .] or numeric values like [x, y, x, y, . . .].
- **fill** – Fill color - can be `black` or `white`.
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

polygon (*xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, outline: Optional[str] = None, clear: bool = False*)
Draw a polygon on the canvas.

Parameters

- **xy** – Sequence of either 2-tuples like [(x, y), (x, y), . . .] or numeric values like [x, y, x, y, . . .].
- **fill** – Fill color - can be `black` or `white`.
- **outline** – Outline color - can be `black` or `white`.

- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

rectangle(*xy: Union[Tuple[int], List[int], None] = None, fill: Optional[str] = None, outline: Optional[str] = None, width: int = 1, clear: bool = False*)
Draw a rectangle on the canvas.

Parameters

- **xy** – Two points defining the bounding box, either as [(x0, y0), (x1, y1)] or [x0, y0, x1, y1]. Default: bounding box of the device.
- **fill** – Fill color - can be `black` or `white`.
- **outline** – Outline color - can be `black` or `white`.
- **width** – Figure width in pixels (default: 1).
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

text(*text: str, pos: Union[Tuple[int], List[int]] = (0, 0), fill: str = 'white', font: Optional[str] = None, font_size: Optional[int] = None, clear: bool = False*)
Draw text on the canvas.

Parameters

- **text** – Text to be drawn.
- **pos** – Position of the text.
- **fill** – Text color (default: `white`).
- **font** – `font` type override.
- **font_size** – `font_size` override.
- **clear** – Set to True if you want to clear the canvas before writing the text (default: False).

2.72 platypush.plugins.mail

class platypush.plugins.mail.**Mail**(*id: int, date: datetime.datetime, size: int, from_: Union[Dict[str, str], List[str], None] = None, to: Union[Dict[str, str], List[str], None] = None, cc: Union[Dict[str, str], List[str], None] = None, bcc: Union[Dict[str, str], List[str], None] = None, subject: str = '', payload: Optional[Any] = None, **kwargs*)

__init__(*id: int, date: datetime.datetime, size: int, from_: Union[Dict[str, str], List[str], None] = None, to: Union[Dict[str, str], List[str], None] = None, cc: Union[Dict[str, str], List[str], None] = None, bcc: Union[Dict[str, str], List[str], None] = None, subject: str = '', payload: Optional[Any] = None, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.plugins.mail.**MailInPlugin**(***kwargs*)
Base class for mail in plugins.

class platypush.plugins.mail.**MailOutPlugin**(***kwargs*)
Base class for mail out plugins.

send(*to: Union[str, List[str]], from_: Optional[str] = None, cc: Union[str, List[str], None] = None, bcc: Union[str, List[str], None] = None, subject: str = ”, body: str = ”, body_type: str = 'plain', attachments: Optional[List[str]] = None, headers: Optional[Dict[str, str]] = None, **connect_args*)

Send an email through the specified SMTP sender.

Parameters

- **to** – Receiver(s), as comma-separated strings or list.
- **from** – Sender email address (`from` is also supported outside of Python contexts).
- **cc** – Carbon-copy addresses, as comma-separated strings or list
- **bcc** – Blind carbon-copy addresses, as comma-separated strings or list
- **subject** – Mail subject.
- **body** – Mail body.
- **body_type** – Mail body type, as a subtype of `text/` (e.g. `html`) (default: `plain`).
- **attachments** – List of attachment files to send.
- **headers** – Key-value map of headers to be added.
- **connect_args** – Parameters for `.connect()`, if you want to override the default server configuration.

class platypush.plugins.mail.**MailPlugin**(***kwargs*)

Base class for mail plugins.

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class platypush.plugins.mail.**ServerInfo**(*server: str, port: int, username: Union[str, NoneType], password: Union[str, NoneType], ssl: bool, keyfile: Union[str, NoneType], certfile: Union[str, NoneType], access_token: Union[str, NoneType], oauth_mechanism: Union[str, NoneType], oauth_vendor: Union[str, NoneType], timeout: Union[int, NoneType]*)

__init__(*server: str, port: int, username: Optional[str], password: Optional[str], ssl: bool, keyfile: Optional[str], certfile: Optional[str], access_token: Optional[str], oauth_mechanism: Optional[str], oauth_vendor: Optional[str], timeout: Optional[int]*) → None

2.73 `platypush.plugins.mail.imap`

class `platypush.plugins.mail.imap.`**`MailImapPlugin`**(*server: str, port: Optional[int] = None, username: Optional[str] = None, password: Optional[str] = None, password_cmd: Optional[str] = None, access_token: Optional[str] = None, oauth_mechanism: Optional[str] = 'XOAUTH2', oauth_vendor: Optional[str] = None, ssl: bool = False, keyfile: Optional[str] = None, certfile: Optional[str] = None, timeout: Optional[int] = 60, **kwargs*)

Plugin to interact with a mail server over IMAP.

Requires:

- **imapclient** (`pip install imapclient`)

`__init__`(*server: str, port: Optional[int] = None, username: Optional[str] = None, password: Optional[str] = None, password_cmd: Optional[str] = None, access_token: Optional[str] = None, oauth_mechanism: Optional[str] = 'XOAUTH2', oauth_vendor: Optional[str] = None, ssl: bool = False, keyfile: Optional[str] = None, certfile: Optional[str] = None, timeout: Optional[int] = 60, **kwargs*)

Parameters

- **`server`** – Server name/address.
- **`port`** – Port (default: 143 for plain, 993 for SSL).
- **`username`** – IMAP username.
- **`password`** – IMAP password.
- **`password_cmd`** – If you don't want to input your password in the configuration, run this command to fetch or decrypt the password.
- **`access_token`** – OAuth2 access token if the server supports OAuth authentication.
- **`oauth_mechanism`** – OAuth2 mechanism (default: `XOAUTH2`).
- **`oauth_vendor`** – OAuth2 vendor (default: None).
- **`ssl`** – Use SSL (default: False).
- **`keyfile`** – Private key file for SSL connection if client authentication is required.
- **`certfile`** – SSL certificate file or chain.
- **`timeout`** – Server connect/read timeout in seconds (default: 60).

`add_flags`(*messages: List[int], flags: Union[str, List[str]], folder: str = 'INBOX', **connect_args*)

Add a set of flags to the specified set of message IDs.

Parameters

- **`messages`** – List of message IDs.
- **`flags`** – List of flags to be added. Examples:

```
['Flagged']
['Seen', 'Deleted']
['Junk']
```

- **folder** – IMAP folder (default: `INBOX`).
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

copy_messages (*messages: List[int], dest_folder: str, source_folder: str = 'INBOX', **connect_args*)

Copy a set of messages IDs from a folder to another.

Parameters

- **messages** – List of message IDs.
- **source_folder** – Source folder.
- **dest_folder** – Destination folder.
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

create_folder (*folder: str*, ***connect_args*)

Create a folder on the server.

Parameters

- **folder** – Folder name.
- **connect_args** –
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

delete_folder (*folder: str*, ***connect_args*)

Delete a folder from the server.

Parameters

- **folder** – Folder name.
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

delete_messages (*messages: List[int], folder: str = 'INBOX', expunge: bool = True, **connect_args*)

Set a specified set of message IDs as deleted.

Parameters

- **messages** – List of message IDs.
- **folder** – IMAP folder (default: `INBOX`).
- **expunge** – If set then the messages will also be expunged from the folder, otherwise they will only be marked as deleted (default: `True`).
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

expunge_messages (*folder: str = 'INBOX'*, *messages: Optional[List[int]] = None*, ***connect_args*)

When `messages` is not set, remove all the messages from `folder` marked as `Deleted`.

Parameters

- **`folder`** – IMAP folder (default: `INBOX`).
- **`messages`** – List of message IDs to expunge (default: all those marked as `Deleted`).
- **`connect_args`** – Arguments to pass to `_get_server_info()` for server configuration override.

`flag_message` (*message: int, folder: str = 'INBOX', **connect_args*)

Add a flag/star to the specified set of message ID.

Parameters

- **`message`** – Message ID.
- **`folder`** – IMAP folder (default: `INBOX`).
- **`connect_args`** – Arguments to pass to `_get_server_info()` for server configuration override.

`flag_messages` (*messages: List[int], folder: str = 'INBOX', **connect_args*)

Add a flag/star to the specified set of message IDs.

Parameters

- **`messages`** – List of message IDs.
- **`folder`** – IMAP folder (default: `INBOX`).
- **`connect_args`** – Arguments to pass to `_get_server_info()` for server configuration override.

`get_folders` (*folder: str = '', pattern: str = '*', **connect_args*) → List[Dict[str, str]]

Get the list of all the folders hosted on the server or those matching a pattern.

Parameters

- **`folder`** – Base folder (default: root).
- **`pattern`** – Pattern to search (default: None).
- **`connect_args`** – Arguments to pass to `_get_server_info()` for server configuration override.

Returns Example:

```
[
    {
        "name": "INBOX",
        "flags": "\Noinferiors",
        "delimiter": "/"
    },
    {
        "name": "Archive",
        "flags": "\Noinferiors",
        "delimiter": "/"
    },
    {
        "name": "Spam",
        "flags": "\Noinferiors",
        "delimiter": "/"
    }
]
```

get_message (*message: int, folder: str = 'INBOX', **connect_args*) → platypush.plugins.mail.Mail
Get the full content of a message given the ID returned by `search()`.

Parameters

- **message** – Message ID.
- **folder** – Folder name (default: `INBOX`).
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

Returns A message in the same format as `search()`, with an added `payload` attribute containing the body/payload.

get_sub_folders (*folder: str = '', pattern: str = '*', **connect_args*) → List[Dict[str, str]]
Get the list of all the sub-folders hosted on the server or those matching a pattern.

Parameters

- **folder** – Base folder (default: root).
- **pattern** – Pattern to search (default: None).
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

Returns Example:

```
[
    {
        "name": "INBOX",
        "flags": "\Noinferiors",
        "delimiter": "/"
    },
    {
        "name": "Archive",
        "flags": "\Noinferiors",
        "delimiter": "/"
    },
    {
        "name": "Spam",
        "flags": "\Noinferiors",
        "delimiter": "/"
    }
]
```

move_messages (*messages: List[int], dest_folder: str, source_folder: str = 'INBOX', **connect_args*)
Move a set of messages IDs from a folder to another.

Parameters

- **messages** – List of message IDs.
- **source_folder** – Source folder.
- **dest_folder** – Destination folder.
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

remove_flags (*messages: List[int], flags: Union[str, List[str]], folder: str = 'INBOX', **connect_args*)
Remove a set of flags to the specified set of message IDs.

Parameters

- **messages** – List of message IDs.
- **flags** – List of flags to be added. Examples:

```
['Flagged']
['Seen', 'Deleted']
['Junk']
```

- **folder** – IMAP folder (default: `INBOX`).
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

rename_folder (*old_name: str, new_name: str, **connect_args*)
Rename a folder on the server.

Parameters

- **old_name** – Previous name
- **new_name** – New name
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

search (*criteria: Union[str, List[str]] = 'ALL', folder: str = 'INBOX', attributes: Optional[List[str]] = None, **connect_args*) → List[platypush.plugins.mail.Mail]
Search for messages on the server that fit the specified criteria.

Parameters

- **criteria** – It should be a sequence of one or more criteria items. Each criteria item may be either unicode or bytes (default: `ALL`). Example values:

```
['UNSEEN']
['SMALLER', 500]
['NOT', 'DELETED']
['TEXT', 'foo bar', 'FLAGGED', 'SUBJECT', 'baz']
['SINCE', '2020-03-14T12:13:45+00:00']
```

  It is also possible (but not recommended) to pass the combined criteria as a single string. In this case IMAPClient won't perform quoting, allowing lower-level specification of criteria. Examples of this style:

```
'UNSEEN'
'SMALLER 500'
'NOT DELETED'
'TEXT "foo bar" FLAGGED SUBJECT "baz"'
'SINCE 03-Apr-2005'
```

  To support complex search expressions, criteria lists can be nested. The following will match messages that are both not flagged and do not have "foo" in the subject:

```
['NOT', ['SUBJECT', 'foo', 'FLAGGED']]
```

- **folder** – Folder to search (default: `INBOX`).

- **attributes** – Attributes that should be retrieved, according to RFC 3501 (default: `ALL = [FLAGS INTERNALDATE RFC822.SIZE ENVELOPE]`). Note that `BODY` will be ignored if specified here for performance reasons - use `get_message()` if you want to get the full content of a message known its ID from `search()`.
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

Returns List of messages matching the criteria. Example:

```
[
  {
    "id": 702,
    "seq": 671,
    "flags": [
      "nonjunk"
    ],
    "internal_date": "2020-08-30T00:31:52+00:00",
    "size": 2908738,
    "bcc": {},
    "cc": {},
    "date": "2020-08-30T00:31:52+00:00",
    "from": {
      "test123@gmail.com": {
        "name": "A test",
        "route": null,
        "email": "test123@gmail.com"
      }
    },
    "message_id": "<SOMETHING@mail.gmail.com>",
    "in_reply_to": "<SOMETHING@mail.gmail.com>",
    "reply_to": {},
    "sender": {
      "test123@gmail.com": {
        "name": "A test",
        "route": null,
        "email": "test123@gmail.com"
      }
    },
    "subject": "Test email",
    "to": {
      "me@gmail.com": {
        "name": null,
        "route": null,
        "email": "me@gmail.com"
      }
    }
  }
]
```

search_flagged_messages(*folder: str = 'INBOX'*, ***connect_args*) → List[platypush.plugins.mail.Mail]
Shortcut for `search()` that returns only the flagged/starred messages.

search_starred_messages(*folder: str = 'INBOX'*, ***connect_args*) → List[platypush.plugins.mail.Mail]
Shortcut for `search()` that returns only the flagged/starred messages.

search_unseen_messages(*folder: str = 'INBOX', **connect_args*) → List[platypush.plugins.mail.Mail]
Shortcut for `search()` that returns only the unread messages.

set_flags(*messages: List[int], flags: Union[str, List[str]], folder: str = 'INBOX', **connect_args*)
Set a set of flags to the specified set of message IDs.

Parameters

- **messages** – List of message IDs.
- **flags** – List of flags to be added. Examples:

```
['Flagged']
['Seen', 'Deleted']
['Junk']
```

- **folder** – IMAP folder (default: `INBOX`).
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

sort(*folder: str = 'INBOX', sort_criteria: Union[str, List[str]] = 'ARRIVAL', criteria: Union[str, List[str]] = 'ALL', **connect_args*) → List[int]
Return a list of message ids from the currently selected folder, sorted by `sort_criteria` and optionally filtered by `criteria`. Note that SORT is an extension to the IMAP4 standard so it may not be supported by all IMAP servers.

Parameters

- **folder** – Folder to be searched (default: `INBOX`).
- **sort_criteria** – It may be a sequence of strings or a single string. IMAPClient will take care any required conversions. Valid *sort_criteria* values:

```
.. code-block:: python

  ['ARRIVAL']
  ['SUBJECT', 'ARRIVAL']
  'ARRIVAL'
  'REVERSE SIZE'
```

- **criteria** – Optional filter for the messages, as specified in `search()`.
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

Returns A list of message IDs that fit the criteria.

undelete_messages(*messages: List[int], folder: str = 'INBOX', **connect_args*)
Remove the `Deleted` flag from the specified set of message IDs.

Parameters

- **messages** – List of message IDs.
- **folder** – IMAP folder (default: `INBOX`).
- **connect_args** – Arguments to pass to `_get_server_info()` for server configuration override.

unflag_message(*message: int, folder: str = 'INBOX', **connect_args*)
Remove a flag/star from the specified set of message ID.

Parameters

- **`message`** – Message ID.
- **`folder`** – IMAP folder (default: `INBOX`).
- **`connect_args`** – Arguments to pass to `_get_server_info()` for server configuration override.

`unflag_messages`(*messages: List[int], folder: str = 'INBOX', **connect_args*)
Remove a flag/star from the specified set of message IDs.

Parameters

- **`messages`** – List of message IDs.
- **`folder`** – IMAP folder (default: `INBOX`).
- **`connect_args`** – Arguments to pass to `_get_server_info()` for server configuration override.

2.74 `platypush.plugins.mail.smtp`

class `platypush.plugins.mail.smtp.`**`MailSmtpPlugin`**(*server: Optional[str] = None, port: Optional[int] = None, local_hostname: Optional[str] = None, source_address: Optional[List[str]] = None, username: Optional[str] = None, password: Optional[str] = None, password_cmd: Optional[str] = None, access_token: Optional[str] = None, oauth_mechanism: Optional[str] = 'XOAUTH2', oauth_vendor: Optional[str] = None, ssl: bool = False, keyfile: Optional[str] = None, certfile: Optional[str] = None, timeout: Optional[int] = 60, **kwargs*)

Plugin to interact with a mail server over SMTP.

`__init__`(*server: Optional[str] = None, port: Optional[int] = None, local_hostname: Optional[str] = None, source_address: Optional[List[str]] = None, username: Optional[str] = None, password: Optional[str] = None, password_cmd: Optional[str] = None, access_token: Optional[str] = None, oauth_mechanism: Optional[str] = 'XOAUTH2', oauth_vendor: Optional[str] = None, ssl: bool = False, keyfile: Optional[str] = None, certfile: Optional[str] = None, timeout: Optional[int] = 60, **kwargs*)

Parameters

- **`server`** – Server name/address.
- **`port`** – Port (default: 25 for plain, 465 for SSL).
- **`local_hostname`** – If specified, local_hostname is used as the FQDN of the local host in the HELO/EHLO command. Otherwise, the local hostname is found using socket.getfqdn().
- **`source_address`** – The optional source_address parameter allows binding to some specific source address in a machine with multiple network interfaces, and/or to some specific source TCP port. It takes a 2-tuple (host, port), for the socket to bind to

as its source address before connecting. If omitted (or if host or port are '' and/or 0 respectively) the OS default behavior will be used.

- **username** – SMTP username.
- **password** – SMTP password.
- **password_cmd** – If you don't want to input your password in the configuration, run this command to fetch or decrypt the password.
- **access_token** – OAuth2 access token if the server supports OAuth authentication.
- **oauth_mechanism** – OAuth2 mechanism (default: `XOAUTH2`).
- **oauth_vendor** – OAuth2 vendor (default: None).
- **ssl** – Use SSL (default: False).
- **keyfile** – Private key file for SSL connection if client authentication is required.
- **certfile** – SSL certificate file or chain.
- **timeout** – Server connect/read timeout in seconds (default: 60).

2.75 platypush.plugins.media

class platypush.plugins.media.**MediaPlugin**(*media_dirs: Optional[List[str]] = None, download_dir: Optional[str] = None, env: Optional[Dict[str, str]] = None, volume: Union[float, int, None] = None, torrent_plugin: str = 'torrent', *args, **kwargs*)

Generic plugin to interact with a media player.

Requires:

- A media player installed (supported so far: mplayer, vlc, mpv, omxplayer, chromecast)
- **python-libtorrent** (`pip install python-libtorrent`), optional, for torrent support over native library
- *rtorrent* installed - optional, for torrent support over rtorrent
- **youtube-dl** installed on your system (see your distro instructions), optional for YouTube support
- **requests** (`pip install requests`), optional, for local files over HTTP streaming supporting
- **ffmpeg**,optional, to get media files metadata

To start the local media stream service over HTTP you will also need the *platypush.backend.http.HttpBackend* backend enabled.

__init__(*media_dirs: Optional[List[str]] = None, download_dir: Optional[str] = None, env: Optional[Dict[str, str]] = None, volume: Union[float, int, None] = None, torrent_plugin: str = 'torrent', *args, **kwargs*)

Parameters

- **media_dirs** – Directories that will be scanned for media files when a search is performed (default: none)
- **download_dir** – Directory where external resources/torrents will be downloaded (default: ~/Downloads)

- **env** – Environment variables key-values to pass to the player executable (e.g. DISPLAY, XDG_VTNR, PULSE_SINK etc.)
- **volume** – Default volume for the player (default: None, maximum volume).
- **torrent_plugin** – Optional plugin to be used for torrent download. Possible values:
  - `torrent` - native `libtorrent`-based plugin (default)
  - `rtorrent` - torrent support over rtorrent RPC/XML interface (recommended)
  - `webtorrent` - torrent support over webtorrent (unstable)

download(*url*, *filename=None*, *directory=None*)
Download a media URL

Parameters

- **url** – Media URL
- **filename** – Media filename (default: URL filename)
- **directory** – Destination directory (default: download_dir)

Returns The absolute path to the downloaded file

get_media_file_duration(*filename*)
Get the duration of a media file in seconds. Requires ffmpeg

next()
Play the next item in the queue

search(*query*, *types=None*, *queue_results=False*, *autoplay=False*, *search_timeout=60*)
Perform a video search.

Parameters

- **query** (*str*) – Query string, video name or partial name
- **types** (*list*) – Video types to search (default: `["youtube", "file", "torrent"]`)
- **queue_results** (*bool*) – Append the results to the current playing queue (default: False)
- **autoplay** (*bool*) – Play the first result of the search (default: False)
- **search_timeout** (*float*) – Search timeout (default: 60 seconds)

start_streaming(*media*, *subtitles=None*, *download=False*)
Starts streaming local media over the specified HTTP port. The stream will be available to HTTP clients on *http://{this-ip}:{http_backend_port}/media/<media_id>*

Parameters

- **media** (*str*) – Media to stream
- **subtitles** (*str*) – Path or URL to the subtitles track to be used
- **download** (*bool*) – Set to True if you prefer to download the file from the streaming link instead of streaming it

Returns dict containing the streaming URL.Example:

```
{
    "id": "0123456abcdef.mp4",
    "source": "file:///mnt/media/movies/movie.mp4",
    "mime_type": "video/mp4",
    "url": "http://192.168.1.2:8008/media/0123456abcdef.mp4"
}
```

class platypush.plugins.media.**PlayerState**
An enumeration.

2.76 platypush.plugins.media.chromecast

class platypush.plugins.media.chromecast.**MediaChromecastPlugin**(*chromecast=None*, **args*, ***kwargs*)

Plugin to interact with Chromecast devices

Supported formats:

- HTTP media URLs
- YouTube URLs
- Plex (through `media.plex` plugin, experimental)

Requires:

- **pychromecast** (`pip install pychromecast`)

__init__(*chromecast=None*, **args*, ***kwargs*)

Parameters **chromecast** (*str*) – Default Chromecast to cast to if no name is specified

disconnect(*chromecast=None*, *timeout=None*, *blocking=True*)
Disconnect a Chromecast and wait for it to terminate

Parameters

- **chromecast** (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.
- **timeout** (*float*) – Number of seconds to wait for disconnection (default: None: block until termination)
- **blocking** (*bool*) – If set (default), then the code will wait until disconnection, otherwise it will return immediately.

get_chromecasts(*tries=2*, *retry_wait=10*, *timeout=60*, *blocking=True*, *callback=None*)
Get the list of Chromecast devices

Parameters

- **tries** (*int*) – Number of retries (default: 2)
- **retry_wait** (*int*) – Number of seconds between retries (default: 10 seconds)
- **timeout** (*int*) – Timeout before failing the call (default: 60 seconds)
- **blocking** (*bool*) – If true, then the function will block until all the Chromecast devices have been scanned. If false, then the provided callback function will be invoked when a new device is discovered

- **callback** (*func*) – If blocking is false, then you can provide a callback function that will be invoked when a new device is discovered

join (*chromecast=None*, *timeout=None*)
Blocks the thread until the Chromecast connection is terminated.

Parameters

- **chromecast** (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.
- **timeout** (*float*) – Number of seconds to wait for disconnection (default: None: block until termination)

mute (*chromecast=None*)
Toggle the mute status on the Chromecast

Parameters chromecast (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.

play (*resource*, *content_type=None*, *chromecast=None*, *title=None*, *image_url=None*, *autoplay=True*, *current_time=0*, *stream_type='BUFFERED'*, *subtitles=None*, *subtitles_lang='en-US'*, *subtitles_mime='text/vtt'*, *subtitle_id=1*)
Cast media to a visible Chromecast

Parameters

- **resource** (*str*) – Media to cast
- **content_type** (*str*) – Content type as a MIME type string
- **chromecast** (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.
- **title** (*str*) – Optional title
- **image_url** (*str*) – URL of the image to use for the thumbnail
- **autoplay** (*bool*) – Set it to false if you don't want the content to start playing immediately (default: true)
- **current_time** (*int*) – Time to start the playback in seconds (default: 0)
- **stream_type** (*str*) – Type of stream to cast. Can be BUFFERED (default), LIVE or UNKNOWN
- **subtitles** (*str*) – URL of the subtitles to be shown
- **subtitles_lang** (*str*) – Subtitles language (default: en-US)
- **subtitles_mime** (*str*) – Subtitles MIME type (default: text/vtt)
- **subtitle_id** (*int*) – ID of the subtitles to be loaded (default: 1)

quit (*chromecast=None*)
Exits the current app on the Chromecast

Parameters chromecast (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.

reboot (*chromecast=None*)
Reboots the Chromecast

Parameters chromecast (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.

set_volume(*volume*, *chromecast=None*)
Set the Chromecast volume

Parameters

- **volume** (*float*) – Volume to be set, between 0 and 100
- **chromecast** (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.

voldown(*chromecast=None*, *step=10*)
Turn down the Chromecast volume by 10% or step.

Parameters

- **chromecast** (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.
- **step** (*float*) – Volume decrement between 0 and 100 (default: 100%)

volup(*chromecast=None*, *step=10*)
Turn up the Chromecast volume by 10% or step.

Parameters

- **chromecast** (*str*) – Chromecast to cast to. If none is specified, then the default configured Chromecast will be used.
- **step** (*float*) – Volume increment between 0 and 100 (default: 100%)

2.77 platypush.plugins.media.gstreamer

class platypush.plugins.media.gstreamer.**MediaGstreamerPlugin**(*sink: Optional[str] = None*, **args*, ***kwargs*)

Plugin to play media over GStreamer.

Requires:

- **gst-python** (`pip install gst-python`)

__init__(*sink: Optional[str] = None*, **args*, ***kwargs*)

Parameters **sink** – GStreamer audio sink (default: `None`, automatic).

back(*offset=60.0*)
Back by (default: 60) seconds

forward(*offset=60.0*)
Forward by (default: 60) seconds

get_volume() → float
Get the volume.

Returns Volume value between 0 and 100.

is_playing()

Returns True if it's playing, False otherwise

load(*resource*, ***args*)
Load/queue a resource/video to the player (alias for `play()`).

mute()
Toggle mute state

pause()
Toggle the paused state

play(*resource: Optional[str] = None*, ***args*)
Play a resource.

Parameters **resource** – Resource to play - can be a local file or a remote URL

quit()
Stop and quit the player (alias for `stop()`)

seek(*position: float*) → dict
Seek backward/forward by the specified number of seconds.

Parameters **position** – Number of seconds relative to the current cursor.

set_position(*position: float*) → dict
Seek backward/forward to the specified absolute position.

Parameters **position** – Stream position in seconds.

Returns Player state.

set_volume(*volume*)
Set the volume.

Parameters **volume** – Volume value between 0 and 100.

status() → dict
Get the current player state.

stop()
Stop and quit the player (alias for `quit()`)

voldown(*step=10.0*)
Volume down by (default: 10)%

volup(*step=10.0*)
Volume up by (default: 10)%

2.78 platypush.plugins.media.kodi

class platypush.plugins.media.kodi.**MediaKodiPlugin**(*host*, *http_port=8080*, *websocket_port=9090*, *username=None*, *password=None*, ***kwargs*)

Plugin to interact with a Kodi media player instance

Requires:

- **kodi-json** (`pip install kodi-json`)
- **websocket-client** (`pip install websocket-client`), optional, for player events support

__init__(*host*, *http_port=8080*, *websocket_port=9090*, *username=None*, *password=None*, ***kwargs*)

Parameters

- **host** (*str*) – Kodi host name or IP

- **http_port** (*int*) – Kodi JSON RPC web port. Remember to enable "Allow remote control via HTTP" in Kodi service settings -> advanced configuration and "Allow remote control from applications" on this system and, optionally, on other systems if the Kodi server is on another machine
- **websocket_port** (*int*) – Kodi JSON RPC websocket port, used to receive player events
- **username** (*str*) – Kodi username (optional)
- **password** (*str*) – Kodi password (optional)

back(*offset=30*, *player_id=None*, **args*, ***kwargs*)
Move the player execution backward by delta_seconds

Parameters **offset** (*float*) – Backward seek duration (default: 30 seconds)

back_btn(**args*, ***kwargs*)
Simulate a back input event

clean_audio_library(**args*, ***kwargs*)
Clean the audio library

clean_video_library(**args*, ***kwargs*)
Clean the video library

down(**args*, ***kwargs*)
Simulate a down input event

forward(*offset=30*, *player_id=None*, **args*, ***kwargs*)
Move the player execution forward by delta_seconds

Parameters **offset** (*float*) – Forward seek duration (default: 30 seconds)

fullscreen(**args*, ***kwargs*)
Set/unset fullscreen mode

get_active_players()
Get the list of active players

get_albums(**args*, ***kwargs*)
Get the list of albums in the audio library

get_artists(**args*, ***kwargs*)
Get the list of artists in the audio library

get_movies(**args*, ***kwargs*)
Get the list of movies on the Kodi server

get_songs(**args*, ***kwargs*)
Get the list of songs in the audio library

is_muted(**args*, ***kwargs*)
Return the muted status of the application

left(**args*, ***kwargs*)
Simulate a left input event

mute(**args*, ***kwargs*)
Mute/unmute the application

notify(*title*, *message*, **args*, ***kwargs*)
Send a notification to the Kodi UI

pause (*player_id=None*, **args*, ***kwargs*)
Play/pause the current media

play (*resource*, **args*, ***kwargs*)
Open and play the specified file or URL

Parameters **resource** – URL or path to the media to be played

quit (**args*, ***kwargs*)
Quit the application

repeat (*player_id=None*, *repeat=None*, **args*, ***kwargs*)
Set/unset repeat mode

right (**args*, ***kwargs*)
Simulate a right input event

scan_audio_library (**args*, ***kwargs*)
Scan the audio library

scan_video_library (**args*, ***kwargs*)
Scan the video library

seek (*position*, *player_id=None*, **args*, ***kwargs*)
Move to the specified time position in seconds

Parameters **position** (`float`) – Seek time in seconds

select (**args*, ***kwargs*)
Simulate a select input event

send_text (*text*, **args*, ***kwargs*)
Simulate a send_text input event

Parameters **text** (`str`) – Text to send

set_position (*position*, *player_id=None*, **args*, ***kwargs*)
Move to the specified time position in seconds

Parameters **position** (`float`) – Seek time in seconds

set_volume (*volume*, **args*, ***kwargs*)
Set the application volume

Parameters **volume** (`int`) – Volume to set between 0 and 100

shuffle (*player_id=None*, *shuffle=None*, **args*, ***kwargs*)
Set/unset shuffle mode

stop (*player_id=None*, **args*, ***kwargs*)
Stop the current media

up (**args*, ***kwargs*)
Simulate an up input event

voldown (*step=10.0*, **args*, ***kwargs*)
Volume down (default: -10%)

volup (*step=10.0*, **args*, ***kwargs*)
Volume up (default: +10%)

2.79 `platypush.plugins.media.mplayer`

class `platypush.plugins.media.mplayer.`**`MediaMplayerPlugin`**(*mplayer_bin=None*, *mplayer_timeout=0.5*, *args=None*, **argv*, ***kwargs*)

Plugin to control MPlayer instances

Requires:

- **mplayer** executable on your system

`__init__`(*mplayer_bin=None*, *mplayer_timeout=0.5*, *args=None*, **argv*, ***kwargs*)
Create the MPlayer wrapper. Note that the plugin methods are populated dynamically by introspecting the mplayer executable. You can verify the supported methods at runtime by using the *list_actions* method.

Parameters

- **`mplayer_bin`** (`str`) – Path to the MPlayer executable (default: search for the first occurrence in your system PATH environment variable)
- **`mplayer_timeout`** (`float`) – Timeout in seconds to wait for more data from MPlayer before considering a response ready (default: 0.5 seconds)
- **`subtitles`** (`str`) – Path to the subtitles file
- **`args`** (`list`) – Default arguments that will be passed to the MPlayer executable

`add_subtitles`(*filename*, ***args*)
Sets media subtitles from filename

`back`(*offset=30.0*)
Back by (default: 30) seconds

`execute`(*cmd*, *args=None*)
Execute a raw MPlayer command. See http://www.mplayerhq.hu/DOCS/tech/slave.txt for a reference or call `platypush.plugins.media.mplayer.list_actions()` to get a list

`forward`(*offset=30.0*)
Forward by (default: 30) seconds

`get_property`(*property*, *args=None*)
Get a player property (e.g. pause, fullscreen etc.). See http://www.mplayerhq.hu/DOCS/tech/slave.txt for a full list of the available properties

`is_playing`()

Returns True if it's playing, False otherwise

`load`(*resource*, *mplayer_args=None*, ***kwargs*)
Load a resource/video in the player.

`mute`()
Toggle mute state

`pause`()
Toggle the paused state

`play`(*resource*, *subtitles=None*, *mplayer_args=None*)
Play a resource.

Parameters

- **`resource`** (`str`) – Resource to play - can be a local file or a remote URL

- **subtitles** (*str*) – Path to optional subtitle file
- **mplayer_args** (*list[str]*) – Extra runtime arguments that will be passed to the MPlayer executable

quit()
Quit the player

remove_subtitles(*index=None*)
Removes the subtitle specified by the index (default: all)

seek(*position*)
Seek backward/forward by the specified number of seconds

Parameters position (*int*) – Number of seconds relative to the current cursor

set_position(*position*)
Seek backward/forward to the specified absolute position

Parameters position (*int*) – Number of seconds from the start

set_property(*property, value, args=None*)
Set a player property (e.g. pause, fullscreen etc.). See http://www.mplayerhq.hu/DOCS/tech/slave.txt for a full list of the available properties

set_volume(*volume*)
Set the volume

Parameters volume (*float*) – Volume value between 0 and 100

status()
Get the current player state.

Returns A dictionary containing the current state.

Example:

```
output = {
    "state": "play"  # or "stop" or "pause"
}
```

step_property(*property, value, args=None*)
Step a player property (e.g. volume, time_pos etc.). See http://www.mplayerhq.hu/DOCS/tech/slave.txt for a full list of the available steppable properties

stop()
Stop the playback

toggle_subtitles()
Toggle the subtitles visibility

voldown(*step=10.0*)
Volume down by (default: 10)%

volup(*step=10.0*)
Volume up by (default: 10)%

2.80 platypush.plugins.media.mpv

class platypush.plugins.media.mpv.**MediaMpvPlugin**(*args=None, *argv, **kwargs*)
Plugin to control MPV instances

Requires:

- **python-mpv** (`pip install python-mpv`)
- **mpv** executable on your system

`__init__`(*args=None*, **argv*, ***kwargs*)
Create the MPV wrapper.

Parameters **`args`** (`dict[str, str]`) – Default arguments that will be passed to the mpv executable as a key-value dict (names without the – prefix). See *man mpv* for available options.

`add_subtitles`(*filename*)
Add a subtitles file

`back`(*offset=30.0*)
Back by (default: 30) seconds

`execute`(*cmd*, ***args*)
Execute a raw mpv command.

`forward`(*offset=30.0*)
Forward by (default: 30) seconds

`get_property`(*property*)
Get a player property (e.g. pause, fullscreen etc.). See `man mpv` for a full list of the available properties

`is_playing`()

Returns True if it's playing, False otherwise

`load`(*resource*, ***args*)
Load/queue a resource/video to the player

`mute`()
Toggle mute state

`next`()
Play the next item in the queue

`pause`()
Toggle the paused state

`play`(*resource*, *subtitles=None*, ***args*)
Play a resource.

Parameters

- **`resource`** (`str`) – Resource to play - can be a local file or a remote URL
- **`subtitles`** (`str`) – Path to optional subtitle file
- **`args`** (`dict[str,str]`) – Extra runtime arguments that will be passed to the mpv executable as a key-value dict (keys without – prefix)

`prev`()
Play the previous item in the queue

`quit`()
Stop and quit the player

`remove_subtitles`()
Removes (hides) the subtitles

seek (*position*)
Seek backward/forward by the specified number of seconds

Parameters **position** (*int*) – Number of seconds relative to the current cursor

set_position (*position*)
Seek backward/forward to the specified absolute position (same as `seek`)

set_property (***props*)
Set the value of an mpv property (e.g. fullscreen, sub_visibility etc.). See `man mpv` for a full list of properties

Parameters **props** (*dict*) – Key-value args for the properties to set

set_subtitles (*filename*, **args*, ***kwargs*)
Sets media subtitles from filename

set_volume (*volume*)
Set the volume

Parameters **volume** (*float*) – Volume value between 0 and 100

status ()
Get the current player state.

Returns A dictionary containing the current state.

Example:

```
output = {
    "filename": "filename or stream URL",
    "state": "play"  # or "stop" or "pause"
}
```

stop ()
Stop and quit the player

toggle_fullscreen ()
Toggle the fullscreen mode

toggle_property (*property*)
Toggle or sets the value of an mpv property (e.g. fullscreen, sub_visibility etc.). See `man mpv` for a full list of properties

Parameters **property** – Property to toggle

toggle_subtitles (*visible=None*)
Toggle the subtitles visibility

voldown (*step=10.0*)
Volume down by (default: 10)%

volup (*step=10.0*)
Volume up by (default: 10)%

2.81 platypush.plugins.media.omxplayer

class platypush.plugins.media.omxplayer.**MediaOmxplayerPlugin** (*args=None*, **argv*, ***kwargs*)
Plugin to control video and media playback using OMXPlayer.

Requires:

- **omxplayer** installed on your system (see your distro instructions)
- **omxplayer-wrapper** (`pip install omxplayer-wrapper`)

`__init__`(*args=None*, **argv*, ***kwargs*)

Parameters **`args`** (*`list`*) – Arguments that will be passed to the OMXPlayer constructor (e.g. subtitles, volume, start position, window size etc.) see https://github.com/popcornmix/omxplayer#synopsis and http://python-omxplayer-wrapper.readthedocs.io/en/latest/omxplayer/#omxplayer.player.OMXPlayer

`back`(*offset=30*)
Back by (default: 30) seconds

`forward`(*offset=30*)
Forward by (default: 30) seconds

`get_volume`() → float

Returns The player volume in percentage [0, 100].

`hide_subtitles`()
Hide the subtitles

`hide_video`()
Hide the video

`is_playing`()

Returns True if it's playing, False otherwise

`load`(*resource*, *pause=False*, ***kwargs*)
Load a resource/video in the player.

Parameters

- **`resource`** (*`str`*) – URL or filename to load
- **`pause`** (*`bool`*) – If set, load the video in paused mode (default: False)

`metadata`()
Get the metadata of the current video

`mute`()
Mute the player

`next`()
Play the next track/video

`pause`()
Pause the playback

`play`(*resource=None*, *subtitles=None*, **args*, ***kwargs*)
Play or resume playing a resource.

Parameters

- **`resource`** – Resource to play. Supported types:
  - Local files (format: `file://<path>/<file>`)
  - Remote videos (format: `https://<url>/<resource>`)
  - YouTube videos (format: `https://www.youtube.com/watch?v=<id>`)
  - Torrents (format: Magnet links, Torrent URLs or local Torrent files)

- **subtitles** – Subtitles file

quit()
Quit the player

seek(*position*)
Seek to the specified number of seconds from the start.

Parameters **position** (*float*) – Number of seconds from the start

set_position(*position*)
Seek to the specified number of seconds from the start (same as *seek()*).

Parameters **position** (*float*) – Number of seconds from the start

set_volume(*volume*)
Set the volume

Parameters **volume** (*float*) – Volume value between 0 and 100

status()
Get the current player state.

Returns A dictionary containing the current state.

Format:

```
output = {
    "duration": Duration in seconds,
    "filename": Media filename,
    "fullscreen": true or false,
    "mute": true or false,
    "path": Media path
    "pause": true or false,
    "position": Position in seconds
    "seekable": true or false
    "state": play, pause or stop
    "title": Media title
    "url": Media url
    "volume": Volume between 0 and 100
    "volume_max": 100,
}
```

stop()
Stop the playback (same as quit)

unmute()
Unmute the player

voldown(*step=10.0*)
Decrease the volume.

Parameters **step** (*float*) – Volume decrease step between 0 and 100 (default: 10%).

volup(*step=10.0*)
Increase the volume.

Parameters **step** (*float*) – Volume increase step between 0 and 100 (default: 10%).

class platypush.plugins.media.omxplayer.**PlayerEvent**
An enumeration.

2.82 `platypush.plugins.media.plex`

class `platypush.plugins.media.plex.`**`MediaPlexPlugin`**(*server*, *username*, *password*, ***kwargs*)

Plugin to interact with a Plex media server

Requires:

- **plexapi** (`pip install plexapi`)

`__init__`(*server*, *username*, *password*, ***kwargs*)

Parameters

- **`server`** (*`str`*) – Plex server name
- **`username`** (*`str`*) – Plex username
- **`password`** – Plex password

`back`(*client*)
Backward playback on a client

`down`(*client*)
Send a down key event to a client

`forward`(*client*)
Forward playback on a client

`get_clients`()
Get the list of active clients

`go_back`(*client*)
Send a back key event to a client

`go_home`(*client*)
Send a home key event to a client

`go_to_media`(*client*)
Send a go to media event to a client

`go_to_music`(*client*)
Send a go to music event to a client

`history`()
Get the history of items played on the server

`left`(*client*)
Send a left key event to a client

`next`(*client*)
Play next item on a client

`next_letter`(*client*)
Send a next letter event to a client

`page_down`(*client*)
Send a page down event to a client

`page_up`(*client*)
Send a page up event to a client

`pause`(*client*)
Send pause event to a client

play (*client=None*, *chromecast=None*, ***kwargs*)
Search and play content on a client or a Chromecast. If no search filter is specified, a play event will be sent to the specified client.

NOTE: Adding and managing play queues through the Plex API isn't fully supported yet, therefore in case multiple items are returned from the search only the first one will be played.

Parameters

- **client** (`str`) – Client name
- **chromecast** (`str`) – Chromecast name
- **kwargs** (`dict`) – Search filter (e.g. title, section, unwatched, director etc.)

playlists ()
Get the playlists on the server

previous (*client*)
Play previous item on a client

random (*client*, *random*)
Set the random status on a client

repeat (*client*, *repeat*)
Set the repeat status on a client

right (*client*)
Send a right key event to a client

search (*section=None*, *title=None*, ***kwargs*)
Return all the items matching the search criteria (default: all library items)

Parameters

- **section** (`str`) – Section to search (Movies, Shows etc.)
- **title** (`str`) – Full or partial title
- **kwargs** (`dict`) – Search criteria - includes e.g. title, unwatched, director, genre etc.

seek (*client*, *offset*)
Send seek event to a client

set_volume (*client*, *volume*)
Set the volume on a client between 0 and 100

stop (*client*)
Send stop event to a client

up (*client*)
Send an up key event to a client

2.83 platypush.plugins.media.subtitles

class platypush.plugins.media.subtitles.**MediaSubtitlesPlugin** (*username*, *password*, *language=None*, ***kwargs*)

Plugin to get video subtitles from OpenSubtitles

Requires:

- **python-opensubtitles** (`pip install -e 'git+https://github.com/agonzalezro/python-opensubtitles#egg=python-opensubtitles'`)
- **webvtt** (`pip install webvtt-py`), optional, to convert srt subtitles into vtt format ready for web streaming
- **requests** (`pip install requests`)

`__init__`(*username*, *password*, *language=None*, ***kwargs*)

Parameters

- **`username`** (*`str`*) – Your OpenSubtitles username
- **`password`** (*`str`*) – Your OpenSubtitles password
- **`language`** (*`str or list[str]`*) – Preferred language name, ISO639 code or OpenSubtitles language ID to be used for the subtitles. Also supports an (ordered) list of preferred languages

`download`(*link*, *media_resource=None*, *path=None*, *convert_to_vtt=False*)

Downloads a subtitle link (.srt/.vtt file or gzip/zip OpenSubtitles archive link) to the specified directory

Parameters

- **`link`** (*`str`*) – Local subtitles file or OpenSubtitles gzip download link
- **`path`** (*`str`*) – Path where the subtitle file will be downloaded (default: temporary file under /tmp)
- **`media_resource`** (*`str`*) – Name of the media resource. If set and if it's a media local file then the subtitles will be saved in the same folder
- **`convert_to_vtt`** (*`bool`*) – If set to True, then the downloaded subtitles will be converted to VTT format (default: no conversion)

Returns dict.

Format:

```
{
    "filename": "/path/to/subtitle/file.srt"
}
```

`get_subtitles`(*resource*, *language=None*)

Get the subtitles data for a video resource

Parameters

- **`resource`** (*`str`*) – Media file, torrent or URL to the media resource
- **`language`** (*`str`*) – Language name or code (default: configured preferred language). Choose 'all' for all the languages

`search`(*resource*, *language=None*)

Alias for *`get_subtitles()`*.

Parameters

- **`resource`** (*`str`*) – Media file, torrent or URL to the media resource
- **`language`** (*`str`*) – Language name or code (default: configured preferred language). Choose 'all' for all the languages

`to_vtt`(*filename*)

Get the VTT content given an SRT file. Will return the original content if the file is already in VTT format.

2.84 `platypush.plugins.media.vlc`

class `platypush.plugins.media.vlc.`**`MediaVlcPlugin`**(*args=None*, *fullscreen=False*, *volume=100*, **argv*, ***kwargs*)

Plugin to control vlc instances

Requires:

- **python-vlc** (`pip install python-vlc`)
- **vlc** executable on your system

`__init__`(*args=None*, *fullscreen=False*, *volume=100*, **argv*, ***kwargs*)

Create the vlc wrapper.

Parameters

- **`args`** (*`list[str]`*) – List of extra arguments to pass to the VLC executable (e.g. `['--sub-language=en', '--snapshot-path=/mnt/snapshots']`)
- **`fullscreen`** (*`bool`*) – Set to True if you want media files to be opened in fullscreen by default (can be overridden by *.play()*) (default: False)
- **`volume`** (*`int`*) – Default media volume (default: 100)

`back`(*offset=30.0*)

Back by (default: 30) seconds

`forward`(*offset=30.0*)

Forward by (default: 30) seconds

`is_playing`()

Returns True if it's playing, False otherwise

`load`(*resource*, ***args*)

Load/queue a resource/video to the player

`mute`()

Toggle mute state

`pause`()

Toggle the paused state

`play`(*resource=None*, *subtitles=None*, *fullscreen=None*, *volume=None*)

Play a resource.

Parameters

- **`resource`** (*`str`*) – Resource to play - can be a local file or a remote URL (default: None == toggle play).
- **`subtitles`** (*`str`*) – Path to optional subtitle file
- **`fullscreen`** (*`bool`*) – Set to explicitly enable/disable fullscreen (default: *fullscreen* configured value or False)
- **`volume`** – Set to explicitly set the playback volume (default: *volume* configured value or 100)

`quit`()

Quit the player (same as *stop*)

`remove_subtitles`()

Removes (hides) the subtitles

seek (*position*)
Seek backward/forward by the specified number of seconds

Parameters **position** (`int`) – Number of seconds relative to the current cursor

set_fullscreen (*fullscreen=True*)
Set fullscreen mode

set_position (*position*)
Seek backward/forward to the specified absolute position (same as `seek`)

set_subtitles (*filename*, ***args*)
Sets media subtitles from filename

set_volume (*volume*)
Set the volume

Parameters **volume** (`float`) – Volume value between 0 and 100

status ()
Get the current player state.

Returns A dictionary containing the current state.

Example:

```
output = {
    "filename": "filename or stream URL",
    "state": "play"  # or "stop" or "pause"
}
```

stop ()
Stop the application (same as *quit*)

toggle_fullscreen ()
Toggle the fullscreen mode

toggle_subtitles (*visibile=None*)
Toggle the subtitles visibility

voldown (*step=10.0*)
Volume down by (default: 10)%

volup (*step=10.0*)
Volume up by (default: 10)%

2.85 platypush.plugins.media.webtorrent

class platypush.plugins.media.webtorrent.**MediaWebtorrentPlugin** (*webtorrent_bin=None*, *webtorrent_port=None*, **args*, ***kwargs*)
Plugin to download and stream videos using webtorrent

Requires:

- **webtorrent** installed on your system (`npm install -g webtorrent`)
- **webtorrent-cli** installed on your system (`npm install -g webtorrent-cli`)

- A media plugin configured for streaming (e.g. media.mplayer, media.vlc, media.mpv or media.omxplayer)

`__init__`(*webtorrent_bin=None*, *webtorrent_port=None*, **args*, ***kwargs*)
media.webtorrent will use the default media player plugin you have configured (e.g. mplayer, omxplayer, mpv) to stream the torrent.

Parameters

- **`webtorrent_bin`** (*`str`*) – Path to your webtorrent executable. If not set, then Platypush will search for the right executable in your PATH
- **`webtorrent_port`** (*`int`*) – Port where the webtorrent will be running streaming server will be running (default: 8000)

`download`(*resource*, ***kwargs*)
Download a media URL

Parameters

- **`url`** – Media URL
- **`filename`** – Media filename (default: URL filename)
- **`directory`** – Destination directory (default: download_dir)

Returns The absolute path to the downloaded file

`load`(*resource*, ***kwargs*)
Load a torrent resource in the player.

`play`(*resource*, *player=None*, *download_only=False*, ***player_args*)
Download and stream a torrent

Parameters

- **`resource`** (*`str`*) – Play a resource, as a magnet link, torrent URL or torrent file path
- **`player`** (*`str`*) – If set, use this plugin type as a player for the torrent. Supported types: 'mplayer', 'vlc', 'omxplayer', 'chromecast', 'mpv'. If not set, then the default configured media plugin will be used.
- **`player_args`** (*`dict`*) – Any arguments to pass to the player plugin's play() method
- **`download_only`** (*`bool`*) – If false then it will start streaming the torrent on the local player once the download starts, otherwise it will just download it (default: false)

`quit`()
Quit the player

`status`()
Get the current player state.

Returns A dictionary containing the current state.

Example:

```
output = {
    "state": "play"  # or "stop" or "pause"
}
```

`stop`()
Stop the playback

class `platypush.plugins.media.webtorrent.`**`TorrentState`**

An enumeration.

2.86 `platypush.plugins.midi`

class `platypush.plugins.midi.`**`MidiPlugin`**(*device_name='Platypush virtual MIDI output'*, **args*, ***kwargs*)

Virtual MIDI controller plugin. It allows you to send custom MIDI messages to any connected devices.

Requires:

- **python-rtmidi** (`pip install python-rtmidi`)

`__init__`(*device_name='Platypush virtual MIDI output'*, **args*, ***kwargs*)

Parameters **`device_name`** (*`str`*) – MIDI virtual device name (default: *Platypush virtual MIDI output*)

`play_note`(*note*, *velocity*, *duration=0*)

Play a note with selected velocity and duration.

Parameters

- **`note`** (*`int`*) – MIDI note in range 0-127 with #60 = C4
- **`velocity`** (*`int`*) – MIDI note velocity in range 0-127
- **`duration`** (*`float`*) – Note duration in seconds. Pass 0 if you don't want the note to get off

`query_ports`()

Returns dict: A list of the available MIDI ports with index and name

`release_all_notes`()

Release all the notes being played.

`release_note`(*note*)

Release a played note.

Parameters **`note`** (*`int`*) – MIDI note in range 0-127 with #60 = C4

`send_message`(*values*, **args*, ***kwargs*)

Parameters **`values`** (*`list[int]`*) – Values is expected to be a list containing the MIDI command code and the command parameters - see reference at https://ccrma.stanford.edu/~craig/articles/linuxmidi/misc/essenmidi.html

Available MIDI commands:

- `0x80` Note Off
- `0x90` Note On
- `0xA0` Aftertouch
- `0xB0` Continuous controller
- `0xC0` Patch change
- `0xD0` Channel Pressure
- `0xE0` Pitch bend

- `0xF0` Start of system exclusive message
- `0xF1` MIDI Time Code Quarter Frame (Sys Common)
- `0xF2` Song Position Pointer (Sys Common)
- `0xF3` Song Select
- `0xF6` Tune Request (Sys Common)
- `0xF7` End of system exclusive message
- `0xF8` Timing Clock (Sys Realtime)
- `0xFA` Start (Sys Realtime)
- `0xFB` Continue (Sys Realtime)
- `0xFC` Stop (Sys Realtime)
- `0xFE` Active Sensing (Sys Realtime)
- `0xFF` System Reset (Sys Realtime)

Parameters

- **`args`** – Extra args that will be passed to `rtmidi.send_message`
- **`kwargs`** – Extra kwargs that will be passed to `rtmidi.send_message`

2.87 `platypush.plugins.ml.cv`

class `platypush.plugins.ml.cv.`**`MlCvPlugin`**(***kwargs*)

Plugin to train and make computer vision predictions using machine learning models.

Requires:

- **numpy** (`pip install numpy`)
- **opencv** (`pip install cv2`)

Also make sure that your OpenCV installation comes with the `dnn` module. To test it:

```
>>> import cv2.dnn
```

`__init__`(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

`predict`(*img*, *model_file*, *classes=None*, *resize=None*, *color_convert=None*)

Make predictions for an input image using a model file. Supported model formats include all the types supported by cv2.dnn (currently supported: Caffe, TensorFlow, Torch, Darknet, DLDT).

Parameters

- **`model_file`** – Path to the model file
- **`img`** – Path to the image
- **`classes`** – List of string labels associated with the output values (e.g. [‘negative’, ‘positive’]). If not set then the index of the output neuron with highest value will be returned.
- **`resize`** – Tuple or list with the resize factor to be applied to the image before being fed to the model (default: None)

- **`color_convert`** – Color conversion to be applied to the image before being fed to the model. It points to a cv2 color conversion constant (e.g. `cv2.COLOR_BGR2GRAY`) and it can be either the constant value itself or a string (e.g. 'COLOR_BGR2GRAY').

2.88 `platypush.plugins.mobile.join`

class `platypush.plugins.mobile.join.`**`InterruptionFilterPolicy`**

An enumeration.

class `platypush.plugins.mobile.join.`**`MobileJoinPlugin`**(*api_key: str*, ***kwargs*)

Control mobile devices and other devices linked to Join (https://joaoapps.com/join/api/). It requires you to have either the Join app installed on an Android device or the Join browser extension installed.

The `device` parameter in the actions can be:

- A device ID or name
- A list or comma-separated list of device IDs/names
- A group name or a list of group names

Supported groups:

- group.all
- group.android
- group.windows10
- group.phone
- group.tablet
- group.pc

`__init__`(*api_key: str*, ***kwargs*)

Parameters **`api_key`** – Join API key. Get your at https://joinjoaomgcd.appspot.com/.

`call_number`(*device*, *number: str*)

Call a phone number through a mobile device connected to Join

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`number`** – Phone number

`find`(*device*)

Make a device ring loudly :param device: Device ID or name, or list of device IDs/names, or group name(s)

`get_devices`()

Returns List of connected devices, each containing 'id', 'name', 'user' and 'has_tasker' attributes

`launch_app`(*device*, *name: str = None*, *package: str = None*)

Launch an app on a device

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)

- **name** – Application name
- **package** – Alternatively to the application name, you can also specify the application package name. You can check the package name for an app by going to its Google Play page and checking the end of the URL. Example: for YouTube this is the URL (https://play.google.com/store/apps/details?id=com.google.android.youtube) and this is the package name (com.google.android.youtube)

push (*device*, *text=None*, *url=None*, *file=None*)
Push a URL or file to one or more devices

Parameters

- **device** – Device ID or name, or list of device IDs/names, or group name(s)
- **text** – Optional description text for the URL or file
- **url** – URL to be pushed
- **file** – A publicly accessible URL of a file. You can also send the url of a file on your personal Google Drive

say (*device*, *text: str*, *language: str = None*)
Say some text through a device's TTS engine

Parameters

- **device** – Device ID or name, or list of device IDs/names, or group name(s)
- **text** – Text to say
- **language** – Language code

send_mms (*device*, *file: str = None*, *text: str = None*, *subject: str = ''*, *number: str = None*, *contact_name: str = None*, *urgent: bool = False*)
Send an MMS through a mobile device connected to Join

Parameters

- **device** – Device ID or name, or list of device IDs/names, or group name(s)
- **text** – Text to be sent
- **number** – Phone number
- **contact_name** – Alternatively to the phone number, you can specify a contact name
- **file** – File attached to the message. Must be a local (to the phone) file or a publicly accessible URL
- **subject** – MMS subject
- **urgent** – Set to True if this is an urgent MMS. This will make the sent message be an MMS instead of an SMS

send_notification (*device*, *title: str = None*, *text: str = None*, *url: str = None*, *file: str = None*, *icon: str = None*, *small_icon: str = None*, *priority: int = 2*, *vibration_pattern=None*, *dismiss_on_touch: bool = False*, *image: str = None*, *group: str = None*, *sound: str = None*, *actions=None*)
Send a notification to a device

Parameters

- **device** – Device ID or name, or list of device IDs/names, or group name(s)
- **title** – Notification title

- **`text`** – Notification text
- **`url`** – URL to be opened on touch
- **`file`** – A publicly accessible URL of a file that will be opened or downloaded on touch. You can also send the url of a file on your personal Google Drive.
- **`icon`** – If a notification is created on the receiving device and this is set, then it'll be used as the notification's icon. If this image has transparency, it'll also be used as the status bar icon is smallicon is not set. It'll also be used to tint the notification with its dominating color
- **`small_icon`** – If a notification is created on the receiving device and this is set, then it'll be used as the notification's status bar icon
- **`priority`** – Control how your notification is displayed: lower priority notifications are usually displayed lower in the notification list. Values from -2 (lowest priority) to 2 (highest priority). Default is 2.
- **`vibration_pattern`** (*`str (comma-separated float values) or list[float]`*) – If the notification is received on an Android device, the vibration pattern in this field will change the way the device vibrates with it. You can easily create a pattern by going to the AutoRemote notification page and generate the pattern in the Vibration Pattern field
- **`dismiss_on_touch`** – If set the notification will be dismissed when touched (default: False)
- **`image`** – Publicly available URL for an image to show up in the notification
- **`group`** – Unique ID to group your notifications with
- **`sound`** – Publicly available URL for a sound to play with the notification
- **`actions`** – Set notification buttons with customized behaviour. This parameter is a list of Join actions configured on the target device that will be mapped to notification input elements. More info on the Joaoapps notifications page

`send_sms`(*device*, *text: str*, *number: str = None*, *contact_name: str = None*)

Send an sms through a mobile device connected to Join

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`text`** – Text to be sent
- **`number`** – Phone number
- **`contact_name`** – Alternatively to the phone number, you can specify a contact name

`set_alarm_volume`(*device*, *volume: float*)

Set alarm volume

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`volume`** – Volume

`set_clipboard`(*device*, *text*)

Write to the clipboard of a device

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`text`** – Text to be set

`set_interruption_filter`(*device*, *policy: str*)
Set interruption filter on one or more devices

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`policy`** – Possible values:
  - 'allow_all': Allow all notifications
  - 'priority_only': Allow only priority notifications and calls
  - 'alarm_only': Allow only alarm-related interruptions
  - 'block_all': Do not allow any interruptions

`set_lock_wallpaper`(*device*, *wallpaper: str*)
Set the lock wallpaper on a device connected to Join

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`wallpaper`** – A publicly accessible URL of an image file. Will set the lockscreen wallpaper on the receiving device if the device has Android 7 or above

`set_media_volume`(*device*, *volume: float*)
Set media volume

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`volume`** – Volume

`set_ring_volume`(*device*, *volume: float*)
Set ring volume

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`volume`** – Volume

`set_wallpaper`(*device*, *wallpaper: str*)
Set the wallpaper on a device connected to Join

Parameters

- **`device`** – Device ID or name, or list of device IDs/names, or group name(s)
- **`wallpaper`** – A publicly accessible URL of an image file. Will set the wallpaper on the receiving device

2.89 `platypush.plugins.mqtt`

class `platypush.plugins.mqtt.`**`MqttPlugin`**(*host=None, port=1883, tls_cafile=None, tls_certfile=None, tls_keyfile=None, tls_version=None, tls_ciphers=None, tls_insecure=False, username=None, password=None, client_id=None, **kwargs*)

This plugin allows you to send custom message to a message queue compatible with the MQTT protocol, see http://mqtt.org/

Requires:

- **paho-mqtt** (`pip install paho-mqtt`)

`__init__`(*host=None, port=1883, tls_cafile=None, tls_certfile=None, tls_keyfile=None, tls_version=None, tls_ciphers=None, tls_insecure=False, username=None, password=None, client_id=None, **kwargs*)

Parameters

- **`host`** (*`str`*) – If set, MQTT messages will by default routed to this host unless overridden in *send_message* (default: None)
- **`port`** (*`int`*) – If a default host is set, specify the listen port (default: 1883)
- **`tls_cafile`** (*`str`*) – If a default host is set and requires TLS/SSL, specify the certificate authority file (default: None)
- **`tls_certfile`** (*`str`*) – If a default host is set and requires TLS/SSL, specify the certificate file (default: None)
- **`tls_keyfile`** (*`str`*) – If a default host is set and requires TLS/SSL, specify the key file (default: None)
- **`tls_version`** (*`str`*) – If TLS/SSL is enabled on the MQTT server and it requires a certain TLS version, specify it here (default: None). Supported versions: `tls` (automatic), `tlsv1`, `tlsv1.1`, `tlsv1.2`.
- **`tls_ciphers`** (*`str`*) – If a default host is set and requires TLS/SSL, specify the supported ciphers (default: None)
- **`tls_insecure`** (*`bool`*) – Set to True to ignore TLS insecure warnings (default: False).
- **`username`** (*`str`*) – If a default host is set and requires user authentication, specify the username ciphers (default: None)
- **`password`** (*`str`*) – If a default host is set and requires user authentication, specify the password ciphers (default: None)
- **`client_id`** (*`str`*) – ID used to identify the client on the MQTT server (default: None). If None is specified then `Config.get('device_id')` will be used.

`publish`(*topic: str, msg: Any, host: Optional[str] = None, port: Optional[int] = None, reply_topic: Optional[str] = None, timeout: int = 60, tls_cafile: Optional[str] = None, tls_certfile: Optional[str] = None, tls_keyfile: Optional[str] = None, tls_version: Optional[str] = None, tls_ciphers: Optional[str] = None, tls_insecure: Optional[bool] = None, username: Optional[str] = None, password: Optional[str] = None*)

Sends a message to a topic.

Parameters

- **`topic`** – Topic/channel where the message will be delivered

- **msg** – Message to be sent. It can be a list, a dict, or a Message object.
- **host** – MQTT broker hostname/IP (default: default host configured on the plugin).
- **port** – MQTT broker port (default: default port configured on the plugin).
- **reply_topic** – If a `reply_topic` is specified, then the action will wait for a response on this topic.
- **timeout** – If `reply_topic` is set, use this parameter to specify the maximum amount of time to wait for a response (default: 60 seconds).
- **tls_cafile** – If TLS/SSL is enabled on the MQTT server and the certificate requires a certificate authority to authenticate it, *ssl_cafile* will point to the provided ca.crt file (default: None).
- **tls_certfile** – If TLS/SSL is enabled on the MQTT server and a client certificate it required, specify it here (default: None).
- **tls_keyfile** – If TLS/SSL is enabled on the MQTT server and a client certificate key it required, specify it here (default: None).
- **tls_version** – If TLS/SSL is enabled on the MQTT server and it requires a certain TLS version, specify it here (default: None). Supported versions: `tls` (automatic), `tlsv1`, `tlsv1.1`, `tlsv1.2`.
- **tls_insecure** – Set to True to ignore TLS insecure warnings (default: False).
- **tls_ciphers** – If TLS/SSL is enabled on the MQTT server and an explicit list of supported ciphers is required, specify it here (default: None).
- **username** – Specify it if the MQTT server requires authentication (default: None).
- **password** – Specify it if the MQTT server requires authentication (default: None).

send_message(**args*, ***kwargs*)
Alias for `platypush.plugins.mqtt.MqttPlugin.publish()`.

2.90 platypush.plugins.music

class platypush.plugins.music.**MusicPlugin**(**args*, ***kwargs*)

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

2.91 platypush.plugins.music.mpd

class platypush.plugins.music.mpd.**MusicMpdPlugin**(*host*, *port=6600*)
This plugin allows you to interact with an MPD/Mopidy music server. MPD (https://www.musicpd.org/) is a flexible server-side protocol/application for handling music collections and playing music, mostly aimed to manage local libraries. Mopidy (https://www.mopidy.com/) is an evolution of MPD, compatible with the original protocol and with support for multiple music sources through plugins (e.g. Spotify, TuneIn, Soundcloud, local files etc.).

NOTE: As of Mopidy 3.0 MPD is an optional interface provided by the `mopidy-mpd` extension. Make sure that you have the extension installed and enabled on your instance to use this plugin with your server.

Requires:

- **python-mpd2** (`pip install python-mpd2`)

`__init__`(*host*, *port=6600*)

Parameters

- **`host`** (*`str`*) – MPD IP/hostname
- **`port`** (*`int`*) – MPD port (default: 6600)

`add`(*resource*, *position=None*)
Add a resource (track, album, artist, folder etc.) to the current playlist

Parameters

- **`resource`** (*`str`*) – Resource path or URI
- **`position`** (*`int`*) – Position where the track(s) will be inserted (default: end of the playlist)

`back`()
Go backward by 15 seconds

`clear`()
Clear the current playlist

`consume`(*value=None*)
Set consume mode

Parameters **`value`** (*`bool`*) – If set, set the consume state this value (true/false). Default: None (toggle current state)

`currentsong`()

Returns The currently played track.

Example response:

```
output = {
    "file": "spotify:track:7CO5ADlDN3DcR2pwlnB14P",
    "time": "255",
    "artist": "Elbow",
    "album": "Little Fictions",
    "title": "Kindling",
    "date": "2017",
    "track": "10",
    "pos": "9",
    "id": "3061",
    "albumartist": "Elbow",
    "x-albumuri": "spotify:album:6q5KhDhf9BZkoob7uAnq19"
}
```

`delete`(*positions*)
Delete the playlist item(s) in the specified position(s).

Parameters **`positions`** (*`list[int]`*) – Positions of the tracks to be removed

Returns The modified playlist

`find`(*filter: dict*, **args*, ***kwargs*)
Find in the database/library by filter.

Parameters **`filter`** – Search filter (e.g. `{"artist": "Led Zeppelin", "album": "IV"}`)

Returns list[dict]

`findadd`(*filter: dict*, **args*, ***kwargs*)
Find in the database/library by filter and add to the current playlist.

Parameters **`filter`** – Search filter (e.g. `{"artist": "Led Zeppelin", "album": "IV"}`)

Returns list[dict]

`forward`()
Go forward by 15 seconds

`listplaylist`(*name*)
List the items in the specified playlist (without metadata)

Parameters **`name`** (*`str`*) – Name of the playlist

`listplaylistinfo`(*name*)
List the items in the specified playlist (with metadata)

Parameters **`name`** (*`str`*) – Name of the playlist

`listplaylists`()

Returns The playlists available on the server as a list of dicts.

Example response:

```
output = [
    {
        "playlist": "Rock",
        "last-modified": "2018-06-25T21:28:19Z"
    },
    {
        "playlist": "Jazz",
        "last-modified": "2018-06-24T22:28:29Z"
    },
    {
        # ...
    }
]
```

`load`(*playlist*, *play=True*)
Load and play a playlist by name

Parameters

- **`playlist`** (*`str`*) – Playlist name
- **`play`** (*`bool`*) – Start playback after loading the playlist (default: True)

`lsinfo`(*uri=None*)
Returns the list of playlists and directories on the server

`move`(*from_pos*, *to_pos*)
Move the playlist item in position <from_pos> to position <to_pos>

Parameters

- **`from_pos`** (*`int`*) – Track current position
- **`to_pos`** (*`int`*) – Track new position

next()
Play the next track

pause()
Pause playback

pause_if_playing()
Pause playback only if it's playing

play(*resource=None*)
Play a resource by path/URI

Parameters **resource** (*str*) – Resource path/URI

play_if_paused()
Play only if it's paused (resume)

play_if_paused_or_stopped()
Play only if it's paused or stopped

play_or_stop()
Play or stop (play state toggle)

play_pos(*pos*)
Play a track in the current playlist by position number

Parameters **pos** – Position number

playid(*track_id*)
Play a track by ID

Parameters **track_id** (*str*) – Track ID

playlistadd(*name*, *uri*)
Add one or multiple resources to a playlist.

Parameters

- **name** (*str*) – Playlist name
- **uri** (*str or list[str]*) – URI or path of the resource(s) to be added

playlistclear(*name*)
Clears all the elements from the specified playlist

Parameters **name** (*str*) – Playlist name

playlistdelete(*name*, *pos*)
Remove one or multiple tracks from a playlist.

Parameters

- **name** (*str*) – Playlist name
- **pos** (*int or list[int]*) – Position or list of positions to remove

playlistinfo()

Returns The tracks in the current playlist as a list of dicts.

Example output:

```
output = [
    {
        "file": "spotify:track:79VtgIoznishPUDWO7Kafu",
        "time": "355",
```

(continues on next page)

(continued from previous page)

```
        "artist": "Elbow",
        "album": "Little Fictions",
        "title": "Trust the Sun",
        "date": "2017",
        "track": "3",
        "pos": "10",
        "id": "3062",
        "albumartist": "Elbow",
        "x-albumuri": "spotify:album:6q5KhDhf9BZkoob7uAnq19"
    },
    {
        "file": "spotify:track:3EzTre0pxmoMYRuhJKMHj6",
        "time": "219",
        "artist": "Elbow",
        "album": "Little Fictions",
        "title": "Gentle Storm",
        "date": "2017",
        "track": "2",
        "pos": "11",
        "id": "3063",
        "albumartist": "Elbow",
        "x-albumuri": "spotify:album:6q5KhDhf9BZkoob7uAnq19"
    },
]
```

playlistmove (*name*, *from_pos*, *to_pos*)
Change the position of a track in the specified playlist

Parameters

- **name** (*str*) – Playlist name
- **from_pos** (*int*) – Original track position
- **to_pos** (*int*) – New track position

plchanges (*version*)
Show what has changed on the current playlist since a specified playlist version number.

Parameters **version** (*int*) – Version number

Returns A list of dicts representing the songs being added since the specified version

previous ()
Play the previous track

random (*value=None*)
Set random mode

Parameters **value** (*bool*) – If set, set the random state this value (true/false). Default: None (toggle current state)

rename (*name*, *new_name*)
Rename a playlist

Parameters

- **name** (*str*) – Original playlist name
- **new_name** – New playlist name

repeat(*value=None*)
Set repeat mode

Parameters **value** (*bool*) – If set, set the repeat state this value (true/false). Default: None (toggle current state)

rm(*playlist*)
Permanently remove playlist(s) by name

Parameters **playlist** (*str or list[str]*) – Name or list of playlist names to remove

save(*name*)
Save the current tracklist to a new playlist with the specified name

Parameters **name** (*str*) – Name of the playlist

search(*filter: dict*, **args*, ***kwargs*)
Free search by filter.

Parameters **filter** – Search filter (e.g. `{"artist": "Led Zeppelin", "album": "IV"}`)

Returns list[dict]

searchadd(*filter*, **args*, ***kwargs*)
Free search by filter and add the results to the current playlist.

Parameters **filter** – Search filter (e.g. `{"artist": "Led Zeppelin", "album": "IV"}`)

Returns list[dict]

searchaddplaylist(*name*)
Search and add a playlist by (partial or full) name

Parameters **name** (*str*) – Playlist name, can be partial

seek(*position*)
Seek to the specified position

Parameters **position** (*int*) – Seek position in seconds, or delta string (e.g. '+15' or '-15') to indicate a seek relative to the current position

seekcur(*value*)
Seek to the specified position (DEPRECATED, use `seek()` instead).

Parameters **value** (*int*) – Seek position in seconds, or delta string (e.g. '+15' or '-15') to indicate a seek relative to the current position

set_volume(*volume*)
Set the volume.

Parameters **volume** (*int*) – Volume value (range: 0-100)

setvol(*vol*)
Set the volume (DEPRECATED, use `set_volume()` instead).

Parameters **vol** (*int*) – Volume value (range: 0-100)

shuffle()
Shuffles the current playlist

single(*value=None*)
Set single mode

Parameters **value** (*bool*) – If set, set the consume state this value (true/false). Default: None (toggle current state)

status()

Returns The current state.

Example response:

```
output = {
    "volume": "9",
    "repeat": "0",
    "random": "0",
    "single": "0",
    "consume": "0",
    "playlist": "52",
    "playlistlength": "14",
    "xfade": "0",
    "state": "play",
    "song": "9",
    "songid": "3061",
    "nextsong": "10",
    "nextsongid": "3062",
    "time": "161:255",
    "elapsed": "161.967",
    "bitrate": "320"
}
```

stop()
Stop playback

voldown(*delta=10*)
Turn down the volume

Parameters **delta** (*int*) – Volume down delta (default: -10%)

volup(*delta=10*)
Turn up the volume

Parameters **delta** (*int*) – Volume up delta (default: +10%)

2.92 platypush.plugins.music.snapcast

class platypush.plugins.music.snapcast.**MusicSnapcastPlugin**(*host='localhost'*, *port=1705*, **args*, ***kwargs*)

Plugin to interact with a [Snapcast](https://github.com/badaix/snapcast) instance, control clients mute status, volume, playback etc.

See https://github.com/badaix/snapcast/blob/master/doc/json_rpc_api/v2_0_0.md for further reference about the returned API types.

__init__(*host='localhost'*, *port=1705*, **args*, ***kwargs*)

Parameters

- **host** (*str*) – Default Snapcast server host (default: localhost)
- **port** (*int*) – Default Snapcast server control port (default: 1705)

delete_client(*client*, *host=None*, *port=None*)
Delete a client from the Snapcast server

Parameters

- **client** (*str*) – Client name or ID
- **host** (*str*) – Snapcast server (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

get_backend_hosts()

Returns A dict with the Snapcast hosts configured on the backend in the format host -> port

get_playing_streams(*exclude_local=False*)
Returns the remote streams configured in the *music.snapcast* backend that are currently active and unmuted.

Parameters **exclude_local** (*bool*) – Exclude localhost connections (default: False)

Returns dict with the host->port mapping.

Example:

```
{
    "hosts": {
        "server_1": 1705,
        "server_2": 1705,
        "server_3": 1705
    }
}
```

group_set_clients(*group*, *clients*, *host=None*, *port=None*)
Sets the clients for a group on a Snapcast server

Parameters

- **group** (*str*) – Group name or ID
- **clients** (*list[str]*) – List of client names or IDs
- **host** (*str*) – Snapcast server (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

group_set_stream(*group*, *stream_id*, *host=None*, *port=None*)
Sets the active stream for a group.

Parameters

- **group** (*str*) – Group name or ID
- **stream_id** (*str*) – Stream ID
- **host** (*str*) – Snapcast server (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

mute(*client=None*, *group=None*, *mute=None*, *host=None*, *port=None*)
Set the mute status of a connected client or group

Parameters

- **client** (*str*) – Client name or ID to mute
- **group** (*str*) – Group ID to mute

- **mute** (*bool*) – Mute status. If not set, the mute status of the selected client/group will be toggled.
- **host** (*str*) – Snapcast server to query (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

set_client_name (*client*, *name*, *host=None*, *port=None*)
Set/change the name of a connected client

Parameters

- **client** (*str*) – Current client name or ID to rename
- **name** (*str*) – New name
- **host** (*str*) – Snapcast server (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

set_group_name (*group*, *name*, *host=None*, *port=None*)
Set/change the name of a group

Parameters

- **group** (*str*) – Group ID to rename
- **name** (*str*) – New name
- **host** (*str*) – Snapcast server (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

set_latency (*client*, *latency*, *host=None*, *port=None*)
Set/change the latency of a connected client

Parameters

- **client** (*str*) – Client name or ID
- **latency** (*float*) – New latency in milliseconds
- **host** (*str*) – Snapcast server (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

status (*host=None*, *port=None*, *client=None*, *group=None*)
Get the status either of a Snapcast server, client or group

Parameters

- **host** (*str*) – Snapcast server to query (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)
- **client** (*str*) – Client ID or name (default: None)
- **group** (*str*) – Group ID or name (default: None)

Returns dict.

Example:

```
"output": {
    "groups": [
        {
        "clients": [
            {
```

(continues on next page)

(continued from previous page)

```
                "config": {
                    "instance": 1,
                    "latency": 0,
                    "name": "",
                    "volume": {
                    "muted": false,
                    "percent": 96
                    }
                },
                "connected": true,
                "host": {
                    "arch": "x86_64",
                    "ip": "YOUR_IP",
                    "mac": "YOUR_MAC",
                    "name": "YOUR_NAME",
                    "os": "YOUR_OS"
                },
                "id": "YOUR_ID",
                "lastSeen": {
                    "sec": 1546648311,
                    "usec": 86011
                },
                "snapclient": {
                    "name": "Snapclient",
                    "protocolVersion": 2,
                    "version": "0.15.0"
                }
                }
            ],
            "id": "YOUR_ID",
            "muted": false,
            "name": "",
            "stream_id": "mopidy"
            }
        ],
        "server": {
            "host": {
            "arch": "armv7l",
            "ip": "",
            "mac": "",
            "name": "YOUR_NAME",
            "os": "YOUR_OS"
            },
            "snapserver": {
            "controlProtocolVersion": 1,
            "name": "Snapserver",
            "protocolVersion": 1,
            "version": "0.15.0"
            }
        },
        "streams": [
            {
                "id": "mopidy",
                "meta": {
                    "STREAM": "mopidy"
                },
                "status": "playing",
```

(continues on next page)

(continued from previous page)

```
                "uri": {
                    "fragment": "",
                    "host": "",
                    "path": "/tmp/snapfifo",
                    "query": {
                    "buffer_ms": "20",
                    "codec": "pcm",
                    "name": "mopidy",
                    "sampleformat": "48000:16:2"
                    },
                    "raw": "pipe:////tmp/snapfifo?buffer_ms=20&codec=pcm&
↪name=mopidy&sampleformat=48000:16:2",
                    "scheme": "pipe"
                }
            }
        ]
}
```

volume (*client*, *volume=None*, *delta=None*, *mute=None*, *host=None*, *port=None*)

Set the volume of a connected client

Parameters

- **client** (*str*) – Client name or ID
- **volume** (*int*) – Absolute volume to set between 0 and 100
- **delta** (*int*) – Relative volume change in percentage (e.g. +10 or -10)
- **mute** (*bool*) – Set to true or false if you want to toggle the muted status
- **host** (*str*) – Snapcast server (default: default configured host)
- **port** (*int*) – Snapcast server port (default: default configured port)

2.93 platypush.plugins.nextcloud

class platypush.plugins.nextcloud.**ClientConfig** (*url: str*, *username: str*, *password: str*)

__init__ (*url: str*, *username: str*, *password: str*) → None

class platypush.plugins.nextcloud.**NextcloudPlugin** (*url: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, ***kwargs*)

Plugin to interact with a NextCloud instance.

Requires:

- **nextcloud-API** (`pip install git+https://github.com/EnterpriseyIntranet/nextcloud-API.git`)

__init__ (*url: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, ***kwargs*)

Parameters

- **url** – URL to the index of your default NextCloud instance.

- **username** – Default NextCloud username.
- **password** – Default NextCloud password.

add_group (*group_id: str*, ***server_args*)
Create a new group.

Parameters

- **group_id** – New group unique ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

add_to_group (*user_id: str*, *group_id: str*, ***server_args*)
Add a user to a group.

Parameters

- **user_id** – User ID/name.
- **group_id** – Group ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

copy (*path: str*, *destination: str*, *user_id: Optional[str] = None*, *overwrite: bool = False*, ***server_args*)
Copy a resource to another path.

Parameters

- **path** – Resource path.
- **destination** – Destination path.
- **user_id** – User ID associated to the resource (default: same as the configured user).
- **overwrite** – Set to `True` if you want to overwrite any existing file (default: `False`).
- **server_args** – Override the default server settings (see `_get_client()` arguments).

create_folder (*path: str*, *user_id: Optional[str] = None*, ***server_args*)
Create a folder.

Parameters

- **path** – Path to the folder.
- **user_id** – User ID associated to the folder (default: same as the configured user).
- **server_args** – Override the default server settings (see `_get_client()` arguments).

create_group_folder (*name: str*, ***server_args*)
Create a new group folder.

Parameters

- **name** – Name/path of the folder.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

create_share(*path: str*, *share_type: str*, *share_with: Optional[str] = None*, *public_upload: bool = False*, *password: Optional[str] = None*, *permissions: Optional[List[str]] = None*, ***server_args*) → dict

Share a file/folder with a user/group or a public link.

Parameters

- **path** – Path to the resource to be shared.
- **share_type** – Share type. Supported values:
  - `user`
  - `group`
  - `public_link`
  - `email`
  - `federated_cloud_share`
  - `circle`
  - `talk_conversation`
- **share_with** – User/group ID, email or conversation ID the resource should be shared with.
- **public_upload** – Whether public upload to the shared folder is allowed (default: False).
- **password** – Optional password to protect the share.
- **permissions** – Share permissions, as a list including any of the following (default: `read`):
  - `read`
  - `update`
  - `create`
  - `delete`
  - `share`
  - `all`
- **server_args** – Override the default server settings (see `_get_client()` arguments).

Returns

The details of the newly created share. Example:

```
{
  "id": "4",
  "share_type": 3,
  "uid_owner": "your_uid",
  "displayname_owner": "Your Name",
  "permissions": 17,
  "can_edit": true,
  "can_delete": true,
  "stime": 1599691325,
  "parent": null,
  "expiration": null,
```

(continues on next page)

(continued from previous page)

```
  "token": "AbCdEfG0123456789",
  "uid_file_owner": "your_uid",
  "note": "",
  "label": "",
  "displayname_file_owner": "Your Name",
  "path": "/Shared Path",
  "item_type": "folder",
  "mimetype": "httpd/unix-directory",
  "storage_id": "home::your-uid",
  "storage": 2,
  "item_source": 13960,
  "file_source": 13960,
  "file_parent": 6,
  "file_target": "/Shared Path",
  "share_with": null,
  "share_with_displayname": "(Shared link)",
  "password": null,
  "send_password_by_talk": false,
  "url": "https://your-domain/nextcloud/index.php/s/
↪AbCdEfG0123456789",
  "mail_send": 1,
  "hide_download": 0
}
```

create_subadmin (*user_id: str*, *group_id: str*, ***server_args*)
Add a user as a subadmin for a group.

Parameters

- **user_id** – User ID/name.
- **group_id** – Group ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

create_user (*user_id: str*, *password: str*, ***server_args*)
Create a user.

Parameters

- **user_id** – User ID/name.
- **password** – User password
- **server_args** – Override the default server settings (see `_get_client()` arguments).

delete_group (*group_id: str*, ***server_args*)
Delete a group.

Parameters

- **group_id** – Group ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

delete_group_folder (*folder_id: Union[int, str]*, ***server_args*)
Delete a new group folder.

Parameters

- **`folder_id`** – Folder ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`delete_notification`(*notification_id: int*, ***server_args*)
Delete a notification.

Parameters

- **`notification_id`** – Notification ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`delete_notifications`(***server_args*)
Delete all notifications for the logged user.

Parameters **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`delete_path`(*path: str*, *user_id: Optional[str] = None*, ***server_args*)
Delete a file or folder.

Parameters

- **`path`** – Path to the resource.
- **`user_id`** – User ID associated to the resource (default: same as the configured user).
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`delete_share`(*share_id: int*, ***server_args*)
Remove the shared state of a resource.

Parameters

- **`share_id`** – Share ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`delete_user`(*user_id: str*, ***server_args*)
Delete a user.

Parameters

- **`user_id`** – User ID/name.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`disable_app`(*app_id: Union[str, int], **server_args*)
Disable an app.

Parameters

- **`app_id`** – App ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`disable_user`(*user_id: str*, ***server_args*)
Disable a user.

Parameters

- **`user_id`** – User ID/name.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`download_file`(*remote_path: str*, *local_path: str*, *user_id: Optional[str] = None*, ***server_args*)
Download a file.

Parameters

- **`remote_path`** – Path to the remote resource.
- **`local_path`** – Path to the local folder.
- **`user_id`** – User ID associated to the resource (default: same as the configured user).
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`edit_user`(*user_id: str, properties: Dict[str, str], **server_args*)
Update a set of properties of a user.

Parameters

- **`user_id`** – User ID/name.
- **`properties`** – Key-value pair of user attributes to be edited.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`enable_app`(*app_id: Union[str, int], **server_args*)
Enable an app.

Parameters

- **`app_id`** – App ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`enable_user`(*user_id: str*, ***server_args*)
Enable a user.

Parameters

- **`user_id`** – User ID/name.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_activities`(*since: Optional[id] = None*, *limit: Optional[int] = None*, *object_type: Optional[str] = None*, *object_id: Optional[int] = None*, *sort: str = 'desc'*, ***server_args*) → List[str]
Get the list of recent activities on an instance.

Parameters

- **`since`** – Only return the activities that have occurred since the specified ID.
- **`limit`** – Maximum number of activities to be returned (default: `None`).
- **`object_type`** – Filter by object type.
- **`object_id`** – Only get the activities related to a specific `object_id`.
- **`sort`** – Sort mode, `asc` for ascending, `desc` for descending (default: `desc`).

- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

Returns The list of selected activities.

`get_app`(*app_id: Union[str, int], **server_args*) → dict
Provides information about an application.

Parameters

- **`app_id`** – App ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_apps`(***server_args*) → List[str]
Get the list of apps installed on a NextCloud instance.

Parameters **`server_args`** – Override the default server settings (see `_get_client()` arguments).

Returns The list of installed apps as strings.

`get_capabilities`(***server_args*) → dict
Returns the capabilities of the server.

Parameters **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_group`(*group_id: str*, ***server_args*) → dict
Get the information of a group.

Parameters

- **`group_id`** – Group ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_group_folder`(*folder_id: Union[int, str], **server_args*) → dict
Get a new group folder.

Parameters

- **`folder_id`** – Folder ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_group_folders`(***server_args*) → list
Get the list new group folder.

Parameters **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_groups`(*search: Optional[str] = None*, *limit: Optional[int] = None*, *offset: Optional[int] = None*, ***server_args*) → List[str]
Search for groups.

Parameters

- **`search`** – Search for groups matching the specified substring.
- **`limit`** – Maximum number of returned entries.
- **`offset`** – Start offset.

- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_notification`(*notification_id: int*, ***server_args*) → Union[dict, str]
Get the content of a notification.

Parameters

- **`notification_id`** – Notification ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_notifications`(***server_args*) → list
Get the list of notifications for the logged user.

Parameters **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_share`(*share_id: int*, ***server_args*) → dict
Get the information of a shared resource.

Parameters

- **`share_id`** – Share ID.
- **`server_args`** – Override the default server settings (see `_get_client()` arguments).

`get_shares`(***server_args*) → List[dict]
Get the list of shares available on the server.

Parameters **`server_args`** – Override the default server settings (see `_get_client()` arguments).

Returns

List of available shares. Example:

```
[
    {
      "id": "4",
      "share_type": 3,
      "uid_owner": "your_uid",
      "displayname_owner": "Your Name",
      "permissions": 17,
      "can_edit": true,
      "can_delete": true,
      "stime": 1599691325,
      "parent": null,
      "expiration": null,
      "token": "AbCdEfG0123456789",
      "uid_file_owner": "your_uid",
      "note": "",
      "label": "",
      "displayname_file_owner": "Your Name",
      "path": "/Shared Path",
      "item_type": "folder",
      "mimetype": "httpd/unix-directory",
      "storage_id": "home::your-uid",
      "storage": 2,
      "item_source": 13960,
```

(continues on next page)

(continued from previous page)

```
        "file_source": 13960,
        "file_parent": 6,
        "file_target": "/Shared Path",
        "share_with": null,
        "share_with_displayname": "(Shared link)",
        "password": null,
        "send_password_by_talk": false,
        "url": "https://your-domain/nextcloud/index.php/s/
↪AbCdEfG0123456789",
        "mail_send": 1,
        "hide_download": 0
    }
]
```

get_subadmin_groups(*user_id: str*, ***server_args*) → List[str]

Get the groups where a given user is subadmin.

Parameters

- **user_id** – User ID/name.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

Returns List of matched groups as strings.

get_user(*user_id: str*, ***server_args*) → dict

Get the details of a user.

Parameters

- **user_id** – User ID/name.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

Returns

User details. Example:

```
{
  "enabled": true,
  "storageLocation": "/mnt/hd/nextcloud/user",
  "id": "user",
  "lastLogin": 1599693750000,
  "backend": "Database",
  "subadmin": [],
  "quota": {
    "free": 6869434515456,
    "used": 1836924441,
    "total": 6871271439897,
    "relative": 0.03,
    "quota": -3
  },
  "email": "info@yourdomain.com",
  "displayname": "Your Name",
  "phone": "+1234567890",
  "address": "",
  "website": "https://yourdomain.com",
  "twitter": "@You",
```

(continues on next page)

(continued from previous page)

```
  "groups": [
    "admin"
  ],
  "language": "en",
  "locale": "",
  "backendCapabilities": {
    "setDisplayName": true,
    "setPassword": true
  }
}
```

get_users (*search: Optional[str] = None*, *limit: Optional[int] = None*, *offset: Optional[int] = None*, ***server_args*) → List[str]

Get the list of users matching some search criteria.

Parameters

- **search** – Return users matching the provided string.
- **limit** – Maximum number of results to be returned (default: no limit).
- **offset** – Search results offset (default: None).

Returns List of the matched user IDs.

grant_access_to_group_folder (*folder_id: Union[int, str], group_id: str, **server_args*)

Grant access to a group folder to a given group.

Parameters

- **folder_id** – Folder ID.
- **group_id** – Group ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

list (*path: str*, *user_id: Optional[str] = None*, *depth: int = 1*, *all_properties: bool = False*, ***server_args*) → List[dict]

List the content of a folder on the NextCloud instance.

Parameters

- **path** – Remote path.
- **user_id** – User ID associated to the resource (default: same as the configured user).
- **depth** – Search depth (default: 1).
- **all_properties** – Return all the file properties available (default: `False`).
- **server_args** – Override the default server settings (see `_get_client()` arguments).

list_favorites (*path: Optional[str] = None*, *user_id: Optional[str] = None*, ***server_args*) → List[dict]

List the favorite items for a user.

Parameters

- **path** – Return only the favorites under this path.
- **user_id** – User ID associated to the resource (default: same as the configured user).

- **server_args** – Override the default server settings (see `_get_client()` arguments).

mark_favorite(*path: Optional[str] = None*, *user_id: Optional[str] = None*, ***server_args*)
Add a path to a user's favorites.

Parameters

- **path** – Resource path.
- **user_id** – User ID associated to the resource (default: same as the configured user).
- **server_args** – Override the default server settings (see `_get_client()` arguments).

move(*path: str*, *destination: str*, *user_id: Optional[str] = None*, *overwrite: bool = False*, ***server_args*)
Move a resource to another path.

Parameters

- **path** – Resource path.
- **destination** – Destination path.
- **user_id** – User ID associated to the resource (default: same as the configured user).
- **overwrite** – Set to `True` if you want to overwrite any existing file (default: `False`).
- **server_args** – Override the default server settings (see `_get_client()` arguments).

remove_from_group(*user_id: str*, *group_id: str*, ***server_args*)
Remove a user from a group.

Parameters

- **user_id** – User ID/name.
- **group_id** – Group ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

remove_subadmin(*user_id: str*, *group_id: str*, ***server_args*)
Remove a user as a subadmin from a group.

Parameters

- **user_id** – User ID/name.
- **group_id** – Group ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

rename_group_folder(*folder_id: Union[int, str], new_name: str, **server_args*)
Rename a group folder.

Parameters

- **folder_id** – Folder ID.
- **new_name** – New folder name.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

revoke_access_to_group_folder(*folder_id: Union[int, str], group_id: str, **server_args*)
Revoke access to a group folder to a given group.

Parameters

- **folder_id** – Folder ID.
- **group_id** – Group ID.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

set_group_folder_permissions(*folder_id: Union[int, str], group_id: str, permissions: List[str], **server_args*)
Set the permissions on a folder for a group.

Parameters

- **folder_id** – Folder ID.
- **group_id** – Group ID.
- **permissions** – New permissions, as a list including any of the following:
  - `read`
  - `update`
  - `create`
  - `delete`
  - `share`
  - `all`
- **server_args** – Override the default server settings (see `_get_client()` arguments).

set_group_folder_quota(*folder_id: Union[int, str], quota: Optional[int], **server_args*)
Set the quota of a group folder.

Parameters

- **folder_id** – Folder ID.
- **quota** – Quota in bytes - set None for unlimited.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

update_share(*share_id: int, public_upload: Optional[bool] = None, password: Optional[str] = None, permissions: Optional[List[str]] = None, expire_date: Optional[str] = None, **server_args*)
Update the permissions of a shared resource.

Parameters

- **share_id** – Share ID.
- **public_upload** – Whether public upload to the shared folder is allowed (default: False).
- **password** – Optional password to protect the share.
- **permissions** – Share permissions, as a list including any of the following (default: `read`):

- read
- update
- create
- delete
- share
- all

- **expire_date** – Share expiration date, in the format `YYYY-MM-DD`.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

upload_file(*remote_path: str, local_path: Optional[str] = None, content: Optional[str] = None, user_id: Optional[str] = None, timestamp: Union[datetime.datetime, int, str, None] = None, **server_args*)

Upload a file.

Parameters

- **remote_path** – Path to the remote resource.
- **local_path** – If set, identifies the path to the local file to be uploaded.
- **content** – If set, create a new file with this content instead of uploading an existing file.
- **user_id** – User ID associated to the resource (default: same as the configured user).
- **timestamp** – File timestamp. If not specified it will be retrieved from the file info or set to `now` if `content` is specified.
- **server_args** – Override the default server settings (see `_get_client()` arguments).

class platypush.plugins.nextcloud.**Permission**

An enumeration.

class platypush.plugins.nextcloud.**ShareType**

An enumeration.

2.94 platypush.plugins.nmap

class platypush.plugins.nmap.**NmapPlugin**(***kwargs*)

Nmap network scanner/mapper integration.

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

scan(*hosts: str, ports: str, args: str, sudo: bool = False*) → Dict[str, Any]

Perform a port scan towards a certain host or network.

Parameters

- **hosts** – Host name/IP or IP subnet to scan (e.g. `192.168.1.0/24`).
- **ports** – Port number, (comma-separated) list or (dash-separated) range to scan (default: all).
- **args** – Additional command line arguments for nmap.

- **sudo** – Execute nmap as root through sudo (default: `False`).

Returns Scan results, as an ip -> host map.

2.95 `platypush.plugins.otp`

class `platypush.plugins.otp.`**`OtpPlugin`**(*secret: Optional[str] = None*, *secret_path: Optional[str] = None*, *provisioning_name: Optional[str] = None*, *issuer_name: Optional[str] = None*, ***kwargs*)

This plugin can be used to generate OTP (One-Time Password) codes compatible with Google Authenticator and other 2FA (Two-Factor Authentication) applications.

Requires:

- **pyotp** (`pip install pyotp`)

`__init__`(*secret: Optional[str] = None*, *secret_path: Optional[str] = None*, *provisioning_name: Optional[str] = None*, *issuer_name: Optional[str] = None*, ***kwargs*)

Parameters

- **secret** – Base32-encoded secret to be used for password generation.
- **secret_path** – If no secret is provided statically, then it will be read from this path (default: `~/.local/share/platypush/otp/secret`). If no secret is found then one will be generated.
- **provisioning_name** – If you want to use the Google Authenticator, you can specify the default email address to associate to your OTPs for the provisioning process here.
- **issuer_name** – If you want to use the Google Authenticator, you can specify the default issuer name to display on your OTPs here.

`get_counter_otp`(*count: int*, *secret: Optional[str] = None*, *secret_path: Optional[str] = None*) → str

Parameters

- **count** – Index for the counter-OTP.
- **secret** – Secret token to be used (overrides configured `secret`).
- **secret_path** – File containing the secret to be used (overrides configured `secret_path`).

Returns A count-based token, as a string.

`get_time_otp`(*secret: Optional[str] = None*, *secret_path: Optional[str] = None*) → str

Parameters

- **secret** – Secret token to be used (overrides configured `secret`).
- **secret_path** – File containing the secret to be used (overrides configured `secret_path`).

Returns A time-based token, as a string.

`provision_counter_otp`(*name: Optional[str] = None*, *issuer_name: Optional[str] = None*, *initial_count=0*, *secret: Optional[str] = None*, *secret_path: Optional[str] = None*) → str

Generate a provisioning URI for a counter-OTP that can be imported in Google Authenticator.

Parameters

- **name** – Name or e-mail address associated to the account used by the Google Authenticator. If None is specified then the value will be read from the configured `provisioning_name`.
- **issuer_name** – Name of the issuer of the OTP (default: default configured `issuer_name` or None).
- **initial_count** – Initial value for the counter (default: 0).
- **secret** – Secret token to be used (overrides configured `secret`).
- **secret_path** – File containing the secret to be used (overrides configured `secret_path`).

Returns Generated provisioning URI.

provision_time_otp(*name: Optional[str] = None*, *issuer_name: Optional[str] = None*, *secret: Optional[str] = None*, *secret_path: Optional[str] = None*) → str

Generate a provisioning URI for a time-OTP that can be imported in Google Authenticator.

Parameters

- **name** – Name or e-mail address associated to the account used by the Google Authenticator. If None is specified then the value will be read from the configured `provisioning_name`.
- **issuer_name** – Name of the issuer of the OTP (default: default configured `issuer_name` or None).
- **secret** – Secret token to be used (overrides configured `secret`).
- **secret_path** – File containing the secret to be used (overrides configured `secret_path`).

Returns Generated provisioning URI.

refresh_secret(*secret_path: Optional[str] = None*) → platypush.message.response.Response

Refresh the secret token for key generation given a secret path.

Parameters **secret_path** – Secret path to refresh (default: default configured path).

verify_counter_otp(*otp: str*, *count: int*, *secret: Optional[str] = None*, *secret_path: Optional[str] = None*) → bool

Verify a code against a stored counter-OTP.

Parameters

- **otp** – Code to be verified.
- **count** – Index for the counter-OTP to be verified.
- **secret** – Secret token to be used (overrides configured `secret`).
- **secret_path** – File containing the secret to be used (overrides configured `secret_path`).

Returns True if the code is valid, False otherwise.

verify_time_otp(*otp: str*, *secret: Optional[str] = None*, *secret_path: Optional[str] = None*) → bool

Verify a code against a stored time-OTP.

Parameters

- **otp** – Code to be verified.

- **secret** – Secret token to be used (overrides configured `secret`).
- **secret_path** – File containing the secret to be used (overrides configured `secret_path`).

Returns True if the code is valid, False otherwise.

2.96 `platypush.plugins.pihole`

class `platypush.plugins.pihole.`**`PiholePlugin`**(*server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = False*, *verify_ssl: bool = True*, ***kwargs*)

Plugin for interacting with a Pi-Hole DNS server for advertisement and content blocking.

`__init__`(*server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = False*, *verify_ssl: bool = True*, ***kwargs*)

Parameters

- **server** – Default Pi-hole server IP address.
- **password** – Password for the default Pi-hole server.
- **api_key** – Alternatively to the password, you can also provide the server `api_key`, as retrieved from http://pi-hole-server/admin/scripts/pi-hole/php/api_token.php
- **ssl** – Set to true if the host uses HTTPS (default: False).
- **verify_ssl** – Set to False to disable SSL certificate check.

`blacklist_add`(*domain: str*, *server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*)

Add a domain to the blacklist.

Parameters

- **domain** – Domain name.
- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

`blacklist_remove`(*domain: str*, *server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*)

Remove a domain from the blacklist.

Parameters

- **domain** – Domain name.
- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

disable(*server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *seconds: Optional[int] = None*, *ssl: bool = None*)
Disable a Pi-hole server.

Parameters

- **seconds** – How long the server will be disabled in seconds (default: None, indefinitely).
- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

enable(*server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*)
Enable a Pi-hole server.

Parameters

- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

get_blacklist(*server: Optional[str] = None*, *ssl: bool = None*) → List[str]
Get the content of the blacklist.

Parameters

- **server** – Server IP address (default: default configured `server` value).
- **ssl** – Set to True if the server uses SSL (default: False).

get_list(*list_name: str*, *server: Optional[str] = None*, *ssl: bool = None*) → List[str]
Get the content of a list stored on the server.

Parameters

- **list_name** – List name
- **server** – Server IP address (default: default configured `server` value).
- **ssl** – Set to True if the server uses SSL (default: False).

get_whitelist(*server: Optional[str] = None*, *ssl: bool = None*) → List[str]
Get the content of the whitelist.

Parameters

- **server** – Server IP address (default: default configured `server` value).
- **ssl** – Set to True if the server uses SSL (default: False).

list_add(*list_name: str*, *domain: str*, *server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*)
Add a domain to a custom list stored on the server.

Parameters

- **list_name** – List name
- **domain** – Domain name.

- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

list_remove(*list_name: str*, *domain: str*, *server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*)
Remove a domain from a custom list stored on the server.

Parameters

- **list_name** – List name
- **domain** – Domain name.
- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

status(*server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*) → platypush.message.response.pihole.PiholeStatusResponse
Get the status and statistics of a running Pi-hole server.

Parameters

- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

Returns *`platypush.message.response.pihole.PiholeStatusResponse`*

whitelist_add(*domain: str*, *server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*)
Add a domain to the whitelist.

Parameters

- **domain** – Domain name.
- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).
- **api_key** – Server API key (default: default configured `api_key` value).
- **ssl** – Set to True if the server uses SSL (default: False).

whitelist_remove(*domain: str*, *server: Optional[str] = None*, *password: Optional[str] = None*, *api_key: Optional[str] = None*, *ssl: bool = None*)
Remove a domain from the whitelist.

Parameters

- **domain** – Domain name.
- **server** – Server IP address (default: default configured `server` value).
- **password** – Server password (default: default configured `password` value).

- **`api_key`** – Server API key (default: default configured `api_key` value).
- **`ssl`** – Set to True if the server uses SSL (default: False).

class `platypush.plugins.pihole.`**`PiholeStatus`**

An enumeration.

2.97 `platypush.plugins.ping`

class `platypush.plugins.ping.`**`PingPlugin`**(*executable: str = 'ping', count: int = 1, timeout: float = 5.0, **kwargs*)

Perform ICMP network ping on remote hosts.

`__init__`(*executable: str = 'ping', count: int = 1, timeout: float = 5.0, **kwargs*)

Parameters

- **`executable`** – Path to the `ping` executable. Default: the first `ping` executable found in PATH.
- **`count`** – Default number of packets that should be sent (default: 1).
- **`timeout`** – Default timeout before failing a ping request (default: 5 seconds).

`ping`(*host: str, count: Optional[int] = None, timeout: Optional[float] = None*) → platypush.message.response.ping.PingResponse

Ping a remote host. :param host: Remote host IP or name :param count: Number of packets that should be sent (default: 1). :param timeout: Timeout before failing a ping request (default: 5 seconds).

2.98 `platypush.plugins.printer.cups`

class `platypush.plugins.printer.cups.`**`PrinterCupsPlugin`**(*host: str = 'localhost', printer: Optional[str] = None, **kwargs*)

A plugin to interact with a CUPS printer server.

Requires:

- **pycups** (`pip install pycups`)

`__init__`(*host: str = 'localhost', printer: Optional[str] = None, **kwargs*)

Parameters

- **`host`** – CUPS host IP/name (default: localhost).
- **`printer`** – Default printer name that should be used.

`accept_jobs`(*printer: Optional[str] = None, host: Optional[str] = None*)

Start accepting jobs on a printer.

Parameters

- **`printer`** – Printer name.
- **`host`** – CUPS server IP/name (default: default configured `host`).

`add_printer`(*name: str, ppd_file: str, info: str, location: Optional[str] = None, host: Optional[str] = None*)

Add a printer.

Parameters

- **`name`** – Printer name - alphanumeric + underscore characters only.
- **`ppd_file`** – Path to the PPD file with the printer information and configuration.
- **`host`** – CUPS server IP/name (default: default configured `host`).
- **`info`** – Human-readable information about the printer.
- **`location`** – Human-readable printer location info.

`add_printer_to_class` (*printer_class: str*, *printer: Optional[str] = None*, *host: Optional[str] = None*)
Add a printer to a class.

Parameters

- **`printer_class`** – Class name.
- **`printer`** – Printer name.
- **`host`** – CUPS server IP/name (default: default configured `host`).

`cancel_job` (*job_id: int*, *purge_job: bool = False*, *host: Optional[str] = None*)
Cancel a printer job.

Parameters

- **`job_id`** – Job ID to cancel.
- **`purge_job`** – Also remove the job from the server (default: False).
- **`host`** – CUPS server IP/name (default: default configured `host`).

`delete_printer` (*printer: str*, *host: Optional[str] = None*)
Delete a printer from a CUPS server.

Parameters

- **`printer`** – Printer name.
- **`host`** – CUPS server IP/name (default: default configured `host`).

`delete_printer_from_class` (*printer_class: str*, *printer: Optional[str] = None*, *host: Optional[str] = None*)
Delete a printer from a class.

Parameters

- **`printer_class`** – Class name.
- **`printer`** – Printer name.
- **`host`** – CUPS server IP/name (default: default configured `host`).

`disable_printer` (*printer: Optional[str] = None*, *host: Optional[str] = None*)
Disable a printer on a CUPS server.

Parameters

- **`printer`** – Printer name.
- **`host`** – CUPS server IP/name (default: default configured `host`).

`enable_printer` (*printer: Optional[str], host: Optional[str] = None*)
Enable a printer on a CUPS server.

Parameters

- **`printer`** – Printer name.
- **`host`** – CUPS server IP/name (default: default configured `host`).

`finish_document` (*printer: Optional[str] = None*, *host: Optional[str] = None*)
Finish sending a document to a printer.

Parameters

- **`printer`** – Printer name (default: default configured `printer`).
- **`host`** – CUPS server IP/name (default: default configured `host`).

`get_classes` (*host: Optional[str] = None*) → Dict[str, Dict[str, Any]]
Get the list of classes on a CUPS server.

Parameters **`host`** – CUPS server IP/name (default: default configured `host`).

Returns dict - class_name -> class_info.

`get_jobs` (*host: Optional[str] = None*) → Dict[int, Dict[str, Any]]
Get the list of active jobs.

Parameters **`host`** – CUPS server IP/name (default: default configured `host`).

Returns A job_id -> job_info dict.

`get_printers` (*host: Optional[str] = None*) → platypush.message.response.printer.cups.PrintersResponse
Get the list of printers registered on a CUPS server. :param host: CUPS server host IP/name (default: default configured `host`). :return: *`platypush.message.response.printer.cups.PrintersResponse`*, as a name -> attributes dict.

`move_job` (*job_id: int*, *source_printer_uri: str*, *target_printer_uri: str*, *host: Optional[str] = None*)
Move a job to another printer/URI.

Parameters

- **`job_id`** – Job ID to cancel.
- **`source_printer_uri`** – Source printer URI.
- **`target_printer_uri`** – Target printer URI.
- **`host`** – CUPS server IP/name (default: default configured `host`).

`print_file` (*filename: str*, *printer: Optional[str] = None*, *host: Optional[str] = None*, *title: Optional[str] = None*, *options: Optional[Dict[str, Any]] = None*) → platypush.message.response.printer.cups.PrinterJobAddedResponse
Print a file.

Parameters

- **`filename`** – Path to the file to print.
- **`printer`** – Printer name (default: default configured `printer`).
- **`host`** – CUPS server IP/name (default: default configured `host`).
- **`title`** – Print title.
- **`options`** – Extra CUPS name->value options.

`print_files` (*filenames: List[str], printer: Optional[str] = None, host: Optional[str] = None, title: Optional[str] = None, options: Optional[Dict[str, Any]] = None*) → platypush.message.response.printer.cups.PrinterJobAddedResponse
Print a list of files.

Parameters

- **`filenames`** – Paths to the files to print.
- **`printer`** – Printer name (default: default configured `printer`).
- **`host`** – CUPS server IP/name (default: default configured `host`).
- **`title`** – Print title.
- **`options`** – Extra CUPS name->value options.

`print_test_page`(*printer: Optional[str] = None*, *host: Optional[str] = None*) → platypush.message.response.printer.cups.PrinterJobAddedResponse

Print the CUPS test page.

Parameters

- **`printer`** – Printer name (default: default configured `printer`).
- **`host`** – CUPS server IP/name (default: default configured `host`).

`reject_jobs`(*printer: Optional[str] = None*, *host: Optional[str] = None*)

Start rejecting jobs on a printer.

Parameters

- **`printer`** – Printer name.
- **`host`** – CUPS server IP/name (default: default configured `host`).

2.99 `platypush.plugins.pushbullet`

class `platypush.plugins.pushbullet.`**`PushbulletPlugin`**(*token: str = None*, ***kwargs*)

This plugin allows you to send pushes and files to your PushBullet account. Note: This plugin will only work if the *`platypush.backend.pushbullet`* backend is configured.

Requires:

- **requests** (`pip install requests`)
- The `platypush.backend.pushbullet.Pushbullet` backend enabled

`__init__`(*token: str = None*, ***kwargs*)

Parameters **`token`** – Pushbullet API token. If not set the plugin will try to retrieve it from the Pushbullet backend configuration, if available

`get_device`(*device*)

Parameters **`device`** – Device ID or name

`get_devices`()

Get the list of available devices

`send_clipboard`(*text: str*)

Copy text to the clipboard of a device.

Parameters **`text`** – Text to be copied.

`send_file`(*filename: str*, *device: str = None*)

Send a file.

Parameters

- **`device`** – Device ID or name (default: None, all devices)

- **filename** – Path to the local file

send_note (*device: str = None*, *body: str = None*, *title: str = None*, *url: str = None*, ***kwargs*)
Send a note push.

Parameters

- **device** – Device ID or name (default: None, all devices)
- **body** – Note body
- **title** – Note title
- **url** – URL attached to the note
- **kwargs** – Push arguments, see https://docs.pushbullet.com/#create-push

2.100 `platypush.plugins.qrcode`

class `platypush.plugins.qrcode.`**QrcodePlugin** (*camera_plugin: Optional[str] = None*, ***kwargs*)
Plugin to generate and scan QR and bar codes.

Requires:

- **numpy** (`pip install numpy`).
- **qrcode** (`pip install 'qrcode[pil]'`) for QR generation.
- **pyzbar** (`pip install pyzbar`) for decoding code from images.
- **Pillow** (`pip install Pillow`) for image management.

__init__ (*camera_plugin: Optional[str] = None*, ***kwargs*)

Parameters **camera_plugin** – Name of the plugin that will be used as a camera to capture images (e.g. `camera.cv` or `camera.pi`).

decode (*image_file: str*) → platypush.message.response.qrcode.QrcodeDecodedResponse
Decode a QR code from an image file.

Parameters **image_file** – Path of the image file.

generate (*content: str*, *output_file: Optional[str] = None*, *show: bool = False*, *format: str = 'png'*, *camera_plugin: Optional[str] = None*) → platypush.message.response.qrcode.QrcodeGeneratedResponse
Generate a QR code. If you configured the *`platypush.backend.http.HttpBackend`* then you can also generate codes directly from the browser through `http://<host>:<port>/qrcode?content=...`.

Parameters

- **content** – Text, URL or content of the QR code.
- **output_file** – If set then the QR code will be exported in the specified image file. Otherwise, a base64-encoded representation of its binary content will be returned in the response as `data`.
- **show** – If True, and if the device where the application runs has an active display, then the generated QR code will be shown on display.
- **format** – Output image format (default: `png`).

- **camera_plugin** – If set then this plugin (e.g. `camera` or `camera.pi`) will be used to capture live images from the camera and search for bar codes or QR-codes.

Returns *platypush.message.response.qrcode.QrcodeGeneratedResponse*.

start_scanning(*camera_plugin: Optional[str] = None*, *duration: Optional[float] = None*, *n_codes: Optional[int] = None*) → Optional[List[platypush.message.response.qrcode.ResultModel]]

Decode QR-codes and bar codes using a camera.

Triggers:

- *platypush.message.event.qrcode.QrcodeScannedEvent* when a code is successfully scanned.

Parameters

- **camera_plugin** – Camera plugin (overrides default `camera_plugin`).
- **duration** – How long the capturing phase should run (default: until `stop_scanning` or app termination).
- **n_codes** – Stop after decoding this number of codes (default: None).

Returns When `duration` or `n_codes` are specified or `stop_scanning` is called, it will return a list of *platypush.message.response.qrcode.ResultModel* instances with the scanned results,

2.101 platypush.plugins.redis

class platypush.plugins.redis.**RedisPlugin**(**args*, ***kwargs*)

Plugin to send messages on Redis queues.

Requires:

- **redis** (`pip install redis`)

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

delete(**args*)

Delete one or multiple keys

Parameters **args** – Keys to delete

expire(*key*, *expiration*)

Set an expiration time in seconds for the specified key

Parameters

- **key** (*str*) – Key to set to expire
- **expiration** (*int*) – Expiration timeout (in seconds)

mget(*keys*, **args*)

Returns The values specified in keys as a key/value dict (wraps MGET)

mset(***kwargs*)

Set key/values based on mapping (wraps MSET)

send_message(*queue*, *msg*, **args*, ***kwargs*)

Send a message to a Redis queue.

Parameters

- **queue** (*str*) – Queue name
- **msg** (*str, bytes, list, dict, Message object*) – Message to be sent
- **args** (*list*) – Args passed to the Redis constructor (see https://redis-py.readthedocs.io/en/latest/#redis.Redis)
- **kwargs** (*dict*) – Kwargs passed to the Redis constructor (see https://redis-py.readthedocs.io/en/latest/#redis.Redis)

2.102 `platypush.plugins.rtorrent`

class `platypush.plugins.rtorrent.`**RtorrentPlugin**(*url: str, poll_seconds: float = 5.0, download_dir: str = '~/.rtorrent/watch', **kwargs*)

Plugin to interact search, download and manage torrents through RTorrent. The usage of this plugin is advised over *platypush.plugins.torrent.TorrentPlugin*, as RTorrent is a more flexible and optimized solution for downloading and managing torrents compared to the Platypush native plugin.

Configuration:

- Install `rtorrent` on your system - on Debian/Ubuntu/Raspbian:

```
apt-get install rtorrent
```

- Configure the `rtorrent` XML/RPC interface, usually by adding the following lines to your `~/.rtorrent.rc`:

```
# Enable XML/RPC
scgi_local = /home/user/.rpc.socket
```

- Use a web server to bridge the RPC interface exposed by RTorrent over HTTP. Some configuration examples are available here. I usually use `lighttpd` because it's easy to configure and it comes with a built-in SCGI module. Install the server e.g. using `apt`:

```
apt-get install lighttpd
```

- Create a base configuration file like this under e.g. `~/.config/rtorrent/lighttpd.conf`:

```
### Base configuration
server.modules = (
    "mod_indexfile",
    "mod_access",
    "mod_alias",
    "mod_redirect",
)

# Make sure that all the directories exist.

# server.document-root isn't really needed, but lighttpd
# won't start if it doesn't find a document root.
server.document-root        = "/home/user/.local/share/rtorrent/html"
server.upload-dirs          = ( "/home/user/.cache/uploads" )
server.errorlog             = "/home/user/.local/log/rtorrent/error.log"
server.pid-file             = "/home/user/.local/run/lighttpd.pid"
```

(continues on next page)

(continued from previous page)

```
server.username             = "your-user"
server.groupname            = "your-group"
server.port                 = 5000

index-file.names            = ( "index.html" )

### Configure the RTorrent XML/RPC endpoint
server.modules += ( "mod_scgi" )
scgi.server = (
    # Bind an endpoint called /RPC2 to your local interface
    "/RPC2" =>
      ( "127.0.0.1" =>
        (
          # Read from the RTorrent XML/RPC socket
          "socket" => "/home/user/.rpc.socket",
          "check-local" => "disable",
          "disable-time" => 0,  # don't disable scgi if connection fails
        )
      )
  )
```

- Start the HTTP service, and optionally enable it as a system/user service:

```
lighttpd -f ~/.config/rtorrent/lighttpd.conf
```

- Start RTorrent and check that the XML/RPC interface works:

```
$ xmlrpc localhost:8000 system.listMethods
# Should return a list with all the methods exposed by RTorrent.
$ xmlrpc localhost:5000 download_list
Result:
Array of 0 items:
```

- It is advised to let the RTorrent instance run in e.g. `screen` or `tmux` on the server machine - it is more reliable than letting the plugin start/stop the instance, and you have an easy CLI interface to attach to manage/monitor your torrents.
- In this example, the URL to configure in the plugin would be `http://localhost:5000/RPC2`.

Triggers:

- `platypush.message.event.torrent.TorrentQueuedEvent` when a new torrent transfer is queued.
- `platypush.message.event.torrent.TorrentRemovedEvent` when a torrent transfer is removed.
- `platypush.message.event.torrent.TorrentDownloadStartEvent` when a torrent transfer starts.
- **`platypush.message.event.torrent.TorrentDownloadedMetadataEvent` when the metadata of a torr**
  transfer has been downloaded.
- `platypush.message.event.torrent.TorrentDownloadProgressEvent` when a transfer is progressing.
- `platypush.message.event.torrent.TorrentPausedEvent` when a transfer is paused.
- `platypush.message.event.torrent.TorrentResumedEvent` when a transfer is resumed.

- *platypush.message.event.torrent.TorrentDownloadCompletedEvent* when a transfer is completed.

`__init__`(*url: str*, *poll_seconds: float = 5.0*, *download_dir: str = '~/.rtorrent/watch'*, ***kwargs*)

Parameters

- **`url`** – HTTP URL that exposes the XML/RPC interface of RTorrent (e.g. `http://localhost:5000/RPC2`).
- **`poll_seconds`** – How often the plugin will monitor for changes in the torrent state (default: 5 seconds).
- **`download_dir`** – Directory where torrents and metadata files will be downloaded (default: `~/.rtorrent/watch`).

`download`(*torrent: str*, *is_media: bool = False*, **_*, ***__*)
Download a torrent.

Parameters

- **`torrent`** – Torrent to download. Supported formats:
  - Magnet URLs
  - Torrent URLs
  - Local torrent files
- **`is_media`** – Set it to true if you're downloading a media file that you'd like to stream as soon as the first chunks are available. If so, then the events and the status method will only include media files

Returns The status of the torrent.

`download_torrent_file`(*torrent: str*) → str
Download a torrent link to `torrent_files_dir`.

Parameters **`torrent`** – Torrent URL, magnet link or local file.

Returns Path to the locally downloaded .torrent file.

`execute`(*method: str*, **args*, ***kwargs*)
Execute a raw command over the RTorrent RPC interface.

Parameters

- **`method`** – Method name.
- **`args`** – Method arguments.
- **`kwargs`** – Method keyword-arguments.

Returns Anything returned by the RPC method.

`list_methods`() → List[str]

Returns The list of methods exposed by the RTorrent instance

`open`(*torrent: str*) → dict
Open a loaded torrent transfer.

Parameters **`torrent`** – Torrent hash.

Returns The status of the torrent.

`pause`(*torrent: str*) → dict
Pause a torrent transfer.

Parameters **`torrent`** – Torrent hash.

Returns The status of the torrent.

`quit`()
Terminate all the active transfers and quit the monitor.

`remove`(*torrent*)
Stop and remove a torrent transfer (without removing the downloaded files).

Parameters **`torrent`** – Torrent hash.

`resume`(*torrent*) → dict
Resume a torrent transfer.

Parameters **`torrent`** – Torrent hash.

Returns The status of the torrent.

`start_monitor`()
Start monitoring the status of the RTorrent instance.

`status`(*torrent: str = None*) → dict
Get the status of the current transfers.

Parameters **`torrent`** – Torrent hash.

Returns

A dictionary:

```
{
  "HASH1234567890": {
    "hash": "HASH1234567890",
    "name": "Your torrent name",
    "save_path": "/home/user/Downloads/Your torrent name",
    "is_active": true,
    "is_open": true,
    "completed_bytes": 666894336,
    "download_rate": 451345,
    "is_multi_file": true,
    "remaining_bytes": 1482827011,
    "size_bytes": 2149721347,
    "load_date": "2020-09-02T18:42:19",
    "peers": 0,
    "state": "paused",
    "start_date": "2020-09-02T18:42:19",
    "finish_date": null,
    "upload_rate": 143967,
    "progress": 31.0,
    "files": ["list", "of", "downloaded", "files"]
  }
}
```

`stop`(*torrent*) → dict
Stop a torrent transfer.

Parameters **`torrent`** – Torrent hash.

Returns The status of the torrent.

`stop_monitor`()
Stop monitoring the status of the RTorrent instance.

2.103 platypush.plugins.sensor

class platypush.plugins.sensor.**SensorPlugin**(***kwargs*)

Sensor abstract plugin. Any plugin that interacts with sensors should implement this class (and the get_measurement() method)

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

get_data(**args*, ***kwargs*)

Alias for `get_measurement`

get_measurement(**args*, ***kwargs*)

Implemented by the subclasses.

Returns

Either a raw scalar:

`output = 273.16`

or a name-value dictionary with the values that have been read:

```
output = {
    "temperature": 21.5,
    "humidity": 41.0
}
```

or a list of values:

```
[
    0.01,
    0.34,
    0.53,
    ...
]
```

2.104 platypush.plugins.serial

class platypush.plugins.serial.**SerialPlugin**(*device=None*, *baud_rate=9600*, ***kwargs*)

The serial plugin can read data from a serial device, as long as the serial device returns a JSON. You can use this plugin to interact for example with some sensors connected through an Arduino. Just make sure that the code on your serial device returns JSON values. If you're using an Arduino or any ATMega compatible device, take a look at https://github.com/bblanchon/ArduinoJson.

__init__(*device=None*, *baud_rate=9600*, ***kwargs*)

Parameters

- **device** (*str*) – Device path (e.g. `/dev/ttyUSB0` or `/dev/ttyACM0`)
- **baud_rate** (*int*) – Serial baud rate (default: 9600)

get_measurement(*device=None*, *baud_rate=None*)

Reads JSON data from the serial device and returns it as a message

Parameters

- **device** (*str*) – Device path (default: default configured device)

- **baud_rate** (*int*) – Baud rate (default: default configured baud_rate)

read (*device=None*, *baud_rate=None*, *size=None*, *end=None*)
Reads raw data from the serial device

Parameters

- **device** (*str*) – Device to read (default: default configured device)
- **baud_rate** (*int*) – Baud rate (default: default configured baud_rate)
- **size** (*int*) – Number of bytes to read
- **end** (*int, bytes or str*) – End of message byte or character

write (*data*, *device=None*, *baud_rate=None*)
Writes data to the serial device.

Parameters

- **device** (*str*) – Device to write (default: default configured device)
- **baud_rate** (*int*) – Baud rate (default: default configured baud_rate)
- **data** (*str, bytes or dict. If dict, it will be serialized as JSON.*) – Data to send to the serial device

2.105 platypush.plugins.shell

class platypush.plugins.shell.**ShellPlugin** (***kwargs*)
Plugin to run custom shell commands.

exec (*cmd*, *background=False*, *ignore_errors=False*)
Execute a command.

Parameters

- **cmd** (*str*) – Command to execute
- **background** – If set to True, execute the process in the background, otherwise wait for the process termination and return its output (deafult: False).
- **ignore_errors** – If set, then any errors in the command execution will be ignored. Otherwise a RuntimeError will be thrown (default value: False)

Returns A response object where the `output` field will contain the command output as a string, and the `errors` field will contain whatever was sent to stderr.

2.106 platypush.plugins.smartthings

class platypush.plugins.smartthings.**SmartthingsPlugin** (*access_token: str*, ***kwargs*)
Plugin to interact with devices and locations registered to a Samsung SmartThings account.

Requires:

- **pysmartthings** (`pip install pysmartthings`)

__init__ (*access_token: str*, ***kwargs*)

Parameters **access_token** – SmartThings API access token - you can get one at https://account.smartthings.com/tokens.

execute(*device: str*, *capability: str*, *command*, *component_id: str = 'main'*, *args: Optional[list] = None*)

Execute a command on a device.

Example request to turn on a device with `switch` capability:

```
{
  "type": "request",
  "action": "smartthings.execute",
  "args": {
    "device": "My Switch",
    "capability": "switch",
    "command": "on"
  }
}
```

Parameters

- **device** – Device ID or name.
- **capability** – Property to be read/written (see device `capabilities` returned from *get_device()*).
- **command** – Command to execute on the `capability` (see https://smartthings.developer.samsung.com/docs/api-ref/capabilities.html).
- **component_id** – ID of the component to execute the command on (default: `main`, i.e. the device itself).
- **args** – Command extra arguments, as a list.

get_device(*device: str*) → dict

Get a device info by ID or name.

Parameters **device** – Device ID or name.

Returns

```
"tv-switch-id": {
    "capabilities": [
        "switch",
        "refresh",
        "healthCheck"
    ],
    "device_id": "tv-switch-id",
    "device_type_id": null,
    "device_type_name": null,
    "device_type_network": null,
    "location_id": "location-id",
    "name": "TV Smart Switch",
    "room_id": "room-1"
}
```

get_location(*location_id: Optional[str] = None*, *name: Optional[str] = None*) → dict

Get the info of a location by ID or name.

```
{
    "name": "My home",
    "location_id": "location-id",
    "country_code": "us",
```

(continues on next page)

(continued from previous page)

```
    "locale": "en-US",
    "latitude": "latitude",
    "longitude": "longitude",
    "temperature_scale": null,
    "region_radius": null,
    "timezone_id": null,
    "rooms": {
        "room-1": {
            "background_image": null,
            "location_id": "location-1",
            "name": "Living Room",
            "room_id": "room-1"
        },
        "room-2": {
            "background_image": null,
            "location_id": "location-1",
            "name": "Bedroom",
            "room_id": "room-2"
        }
    }
}
```

info() → Dict[str, Dict[str, dict]]

Return the objects registered to the account, including locations and devices.

```
{
    "devices": {
        "smart-tv-id": {
            "capabilities": [
                "ocf",
                "switch",
                "audioVolume",
                "audioMute",
                "tvChannel",
                "mediaInputSource",
                "mediaPlayback",
                "mediaTrackControl",
                "custom.error",
                "custom.picturemode",
                "custom.soundmode",
                "custom.accessibility",
                "custom.launchapp",
                "custom.recording",
                "custom.tvsearch",
                "custom.disabledCapabilities",
                "samsungvd.ambient",
                "samsungvd.ambientContent",
                "samsungvd.ambient18",
                "samsungvd.mediaInputSource",
                "refresh",
                "execute",
                "samsungvd.firmwareVersion",
                "samsungvd.supportsPowerOnByOcf"
            ],
            "device_id": "smart-tv-id",
            "device_type_id": null,
```

(continues on next page)

(continued from previous page)

```
            "device_type_name": null,
            "device_type_network": null,
            "location_id": "location-id",
            "name": "Samsung Smart TV",
            "room_id": "room-1"
        },
        "tv-switch-id": {
            "capabilities": [
                "switch",
                "refresh",
                "healthCheck"
            ],
            "device_id": "tv-switch-id",
            "device_type_id": null,
            "device_type_name": null,
            "device_type_network": null,
            "location_id": "location-id",
            "name": "TV Smart Switch",
            "room_id": "room-1"
        },
        "lights-switch-id": {
            "capabilities": [
                "switch",
                "refresh",
                "healthCheck"
            ],
            "device_id": "lights-switch-id",
            "device_type_id": null,
            "device_type_name": null,
            "device_type_network": null,
            "location_id": "location-id",
            "name": "Lights Switch",
            "room_id": "room-2"
        }
    },
    "locations": {
        "location-id": {
            "name": "My home",
            "location_id": "location-id",
            "country_code": "us",
            "locale": "en-US",
            "latitude": "latitude",
            "longitude": "longitude",
            "temperature_scale": null,
            "region_radius": null,
            "timezone_id": null,
            "rooms": {
                "room-1": {
                    "background_image": null,
                    "location_id": "location-1",
                    "name": "Living Room",
                    "room_id": "room-1"
                },
                "room-2": {
                    "background_image": null,
                    "location_id": "location-1",
                    "name": "Bedroom",
```

(continues on next page)

(continued from previous page)

```
                        "room_id": "room-2"
                    }
                }
            }
        }
}
```

off (*device: str*, **args*, ***kwargs*) → dict
Turn off a device with `switch` capability.

Parameters **device** – Device name or ID.

Returns Device status

on (*device: str*, **args*, ***kwargs*) → dict
Turn on a device with `switch` capability.

Parameters **device** – Device name or ID.

Returns Device status

status (*device: Union[str, List[str], None] = None*) → List[dict]
Refresh and return the status of one or more devices.

Parameters **device** – Device or list of devices to refresh (default: all)

Returns

A list containing on entry per device, and each entry containing the current device state. Example:

```
[
  {
    "device_id": "switch-1",
    "name": "Fan",
    "switch": false
  },
  {
    "device_id": "tv-1",
    "name": "Samsung Smart TV",
    "switch": true
  }
]
```

switches

Returns

List of switch devices statuses in ***platypush.plugins.switch.SwitchPlugin*** **compatible format.**
Example:

```
[
  {
    "id": "switch-1",
    "name": "Fan",
    "on": false
  },
  {
    "id": "tv-1",
    "name": "Samsung Smart TV",
```

(continues on next page)

(continued from previous page)

```
    "on": true
  }
]
```

toggle(*device: str*, **args*, ***kwargs*) → dict

Toggle a device with `switch` capability.

Parameters **device** – Device name or ID.

Returns Device status

2.107 `platypush.plugins.sound`

class `platypush.plugins.sound.`**PlaybackState**

An enumeration.

class `platypush.plugins.sound.`**RecordingState**

An enumeration.

class `platypush.plugins.sound.`**SoundPlugin**(*input_device=None*, *output_device=None*, *input_blocksize=1024*, *output_blocksize=1024*, ***kwargs*)

Plugin to interact with a sound device.

Triggers:

- *platypush.message.event.sound.SoundPlaybackStartedEvent* on playback start
- *platypush.message.event.sound.SoundPlaybackStoppedEvent* on playback stop
- *platypush.message.event.sound.SoundPlaybackPausedEvent* on playback pause
- *platypush.message.event.sound.SoundRecordingStartedEvent* on recording start
- *platypush.message.event.sound.SoundRecordingStoppedEvent* on recording stop
- *platypush.message.event.sound.SoundRecordingPausedEvent* on recording pause

Requires:

- **sounddevice** (`pip install sounddevice`)
- **soundfile** (`pip install soundfile`)
- **numpy** (`pip install numpy`)

__init__(*input_device=None*, *output_device=None*, *input_blocksize=1024*, *output_blocksize=1024*, ***kwargs*)

Parameters

- **input_device** (*int or str*) – Index or name of the default input device. Use `platypush.plugins.sound.query_devices()` to get the available devices. Default: system default
- **output_device** (*int or str*) – Index or name of the default output device. Use `platypush.plugins.sound.query_devices()` to get the available devices. Default: system default
- **input_blocksize** (*int*) – Blocksize to be applied to the input device. Try to increase this value if you get input overflow errors while recording. Default: 1024

- **output_blocksize** (*int*) – Blocksize to be applied to the output device. Try to increase this value if you get output underflow errors while playing. Default: 1024

pause_playback (*streams=None*)

Parameters **streams** (*list[int]*) – Streams to pause by index (default: all)

play (*file=None, sound=None, device=None, blocksize=None, bufsize=None, samplerate=None, channels=None, stream_name=None, stream_index=None*)
Plays a sound file (support formats: wav, raw) or a synthetic sound.

Parameters

- **file** (*str*) – Sound file path. Specify this if you want to play a file
- **sound** – Sound to play. Specify this if you want to play synthetic sounds. You can also create polyphonic sounds by just calling play multiple times.
- **device** (*int or str*) – Output device (default: default configured device or system default audio output if not configured)
- **blocksize** (*int*) – Audio block size (default: configured *output_blocksize* or 2048)
- **bufsize** (*int*) – Size of the audio buffer (default: 20 frames for audio files, 2 frames for synth sounds)
- **samplerate** (*int*) – Audio samplerate. Default: audio file samplerate if in file mode, 44100 Hz if in synth mode
- **channels** (*int*) – Number of audio channels. Default: number of channels in the audio file in file mode, 1 if in synth mode
- **stream_index** (*int*) – If specified, play to an already active stream index (you can get them through `platypush.plugins.sound.query_streams()`). Default: creates a new audio stream through PortAudio.
- **stream_name** (*str*) – Name of the stream to play to. If set, the sound will be played to the specified stream name, or a stream with that name will be created. If not set, and `stream_index` is not set either, then a new stream will be created on the next available index and named `platypush-stream-<index>`.

query_devices (*category=None*)
Query the available devices

Parameters **category** (*str*) – Device category to query. Can be either input or output. Default: None (query all devices)

Returns A dictionary representing the available devices.

Example:

```
[
    {
        "name": "pulse",
        "hostapi": 0,
        "max_input_channels": 32,
        "max_output_channels": 32,
        "default_low_input_latency": 0.008684807256235827,
        "default_low_output_latency": 0.008684807256235827,
        "default_high_input_latency": 0.034807256235827665,
        "default_high_output_latency": 0.034807256235827665,
        "default_samplerate": 44100
```

(continues on next page)

(continued from previous page)

```
    },
    {
        "name": "default",
        "hostapi": 0,
        "max_input_channels": 32,
        "max_output_channels": 32,
        "default_low_input_latency": 0.008684807256235827,
        "default_low_output_latency": 0.008684807256235827,
        "default_high_input_latency": 0.034807256235827665,
        "default_high_output_latency": 0.034807256235827665,
        "default_samplerate": 44100
    }
]
```

query_streams()

Returns A list of active audio streams

record(*outfile=None*, *duration=None*, *device=None*, *sample_rate=None*, *format=None*, *blocksize=None*, *latency=0*, *channels=1*, *subtype='PCM_24'*)

Records audio to a sound file (support formats: wav, raw)

Parameters

- **outfile** (*str*) – Sound file (default: the method will create a temporary file with the recording)
- **duration** (*float*) – Recording duration in seconds (default: record until stop event)
- **device** (*int or str*) – Input device (default: default configured device or system default audio input if not configured)
- **sample_rate** (*int*) – Recording sample rate (default: device default rate)
- **format** (*str*) – Audio format (default: WAV)
- **blocksize** (*int*) – Audio block size (default: configured *input_blocksize* or 2048)
- **latency** (*float*) – Device latency in seconds (default: 0)
- **channels** (*int*) – Number of channels (default: 1)
- **subtype** (*str*) – Recording subtype - see Soundfile docs - Subtypes for a list of the available subtypes (default: PCM_24)

recordplay(*duration=None*, *input_device=None*, *output_device=None*, *sample_rate=None*, *blocksize=None*, *latency=0*, *channels=1*, *dtype=None*)

Records audio and plays it on an output sound device (audio pass-through)

Parameters

- **duration** (*float*) – Recording duration in seconds (default: record until stop event)
- **input_device** (*int or str*) – Input device (default: default configured device or system default audio input if not configured)
- **output_device** (*int or str*) – Output device (default: default configured device or system default audio output if not configured)
- **sample_rate** (*int*) – Recording sample rate (default: device default rate)

- **blocksize** (*int*) – Audio block size (default: configured *output_blocksize* or 2048)
- **latency** (*float*) – Device latency in seconds (default: 0)
- **channels** (*int*) – Number of channels (default: 1)
- **dtype** (*str*) – Data type for the recording - see Soundfile docs - Recording for available types (default: input device default)

release(*stream_index=None*, *stream_name=None*, *sound_index=None*, *midi_note=None*, *frequency=None*)

Remove a sound from an active stream, either by sound index (use `platypush.sound.plugin.SoundPlugin.query_streams()` to get the sounds playing on the active streams), midi_note, frequency or absolute file path.

Parameters

- **stream_index** (*str*) – Stream index (default: sound removed from all the active streams)
- **stream_name** – Stream name (default: sound removed from all the active streams)
- **sound_index** (*int*) – Sound index
- **midi_note** (*int*) – MIDI note
- **frequency** (*float*) – Sound frequency

stop_playback(*streams=None*)

Parameters **streams** (*list[int] or list[str]*) – Streams to stop by index or name (default: all)

stream_recording(*device=None*, *fifo=None*, *duration=None*, *sample_rate=None*, *dtype='float32'*, *blocksize=None*, *latency=0*, *channels=1*)

Return audio data from an audio source

Parameters

- **device** (*int or str*) – Input device (default: default configured device or system default audio input if not configured)
- **fifo** (*str*) – Path of the FIFO that will be used to exchange audio samples (default: /tmp/inputstream)
- **duration** (*float*) – Recording duration in seconds (default: record until stop event)
- **sample_rate** (*int*) – Recording sample rate (default: device default rate)
- **dtype** (*str*) – Data type for the audio samples. Supported types: 'float64', 'float32', 'int32', 'int16', 'int8', 'uint8'. Default: float32
- **blocksize** (*int*) – Audio block size (default: configured *input_blocksize* or 2048)
- **latency** (*float*) – Device latency in seconds (default: 0)
- **channels** (*int*) – Number of channels (default: 1)

2.108 `platypush.plugins.ssh`

class `platypush.plugins.ssh.`**`SshPlugin`**(*key_file: Optional[str] = None*, *passphrase: Optional[str] = None*, ***kwargs*)

SSh plugin.

Requires:

- **paramiko** (`pip install paramiko`)

`__init__`(*key_file: Optional[str] = None*, *passphrase: Optional[str] = None*, ***kwargs*)

Parameters

- **`key_file`** – Default key file (default: any "id_rsa", "id_dsa", "id_ecdsa", or "id_ed25519" key discoverable in `~/.ssh/`.
- **`passphrase`** – Key file passphrase (default: None).

`chdir`(*path: str*, *keep_alive: bool = False*, ***kwargs*) → None

Change directory to the specified path.

Parameters

- **`path`** – Destination path.
- **`keep_alive`** – Keep the connection active after running the command (default: False).
- **`kwargs`** – Arguments for *`platypush.plugins.ssh.SshPlugin.connect()`*.

`chmod`(*path: str*, *mode: int*, *keep_alive: bool = False*, ***kwargs*) → None

Change the access rights of a path.

Parameters

- **`path`** – Path to be modified.
- **`mode`** – Access permissions (in octal mode).
- **`keep_alive`** – Keep the connection active after running the command (default: False).
- **`kwargs`** – Arguments for *`platypush.plugins.ssh.SshPlugin.connect()`*.

`chown`(*path: str*, *uid: int*, *gid: int*, *keep_alive: bool = False*, ***kwargs*) → None

Change the owner of a path.

Parameters

- **`path`** – Path to be modified.
- **`uid`** – New user ID.
- **`gid`** – New group ID.
- **`keep_alive`** – Keep the connection active after running the command (default: False).
- **`kwargs`** – Arguments for *`platypush.plugins.ssh.SshPlugin.connect()`*.

connect (*host: str*, *port: int = 22*, *user: Optional[str] = None*, *password: Optional[str] = None*, *key_file: Optional[str] = None*, *passphrase: Optional[str] = None*, *compress: bool = False*, *timeout: Optional[int] = None*, *auth_timeout: Optional[int] = None*) → None

Open an SSH connection.

Parameters

- **host** – Host name or IP. Can also be in the format `[user]@<host>:[port]`.
- **port** – Remote port (default: 22).
- **user** – Username (default: None, same user name as the one running platypush).
- **password** – Password (default: None).
- **key_file** – Key file to use for authentication (default: None).
- **passphrase** – Passphrase for the key file (default: None).
- **compress** – Compress data on the connection (default: False).
- **timeout** – Data transfer timeout in seconds (default: None).
- **auth_timeout** – Authentication timeout in seconds (default: None).

disconnect (*host: str*, *port: int = 22*, *user: Optional[str] = None*) → None

Close a connection to a host.

Parameters

- **host** – Host name or IP. Can also be in the format `[user]@<host>:[port]`.
- **port** – Remote port (default: 22).
- **user** – Username (default: None, same user name as the one running platypush).

exec (*cmd: str*, *keep_alive: bool = False*, *timeout: Optional[int] = None*, *stdin: Optional[str] = None*, *env: Optional[Dict[str, str]] = None*, ***kwargs*) → platypush.message.response.Response

Run a command on a host.

Parameters

- **cmd** – Command to run
- **keep_alive** – Keep the connection active after running the command (default: False).
- **timeout** – Communication timeout in seconds (default: None).
- **stdin** – Optional string to pass on the stdin of the command.
- **env** – Dictionary of environment variables to be used for the connection (default: None).
- **kwargs** – Arguments for `platypush.plugins.ssh.SshPlugin.connect()`.

Returns The output of the executed command.

get (*remote_path: str*, *local_path: str*, *recursive: bool = False*, *keep_alive: bool = False*, ***kwargs*) → None

Download a file or folder from an SSH server.

Parameters

- **remote_path** – Remote path (file or directory).
- **local_path** – Local path (file or directory).

- **`recursive`** – Set to True if you want to recursively download folders (default: False).
- **`keep_alive`** – Keep the connection active after running the command (default: False).
- **`kwargs`** – Arguments for *`platypush.plugins.ssh.SshPlugin.connect()`*.

`getcwd`(*keep_alive: bool = False*, ***kwargs*) → str

Get the current working directory.

Parameters

- **`keep_alive`** – Keep the connection active after running the command (default: False).
- **`kwargs`** – Arguments for *`platypush.plugins.ssh.SshPlugin.connect()`*.

`keygen`(*filename: str*, *type: str = 'rsa'*, *bits: int = 4096*, *comment: Optional[str] = None*, *passphrase: Optional[str] = None*) → platypush.message.response.ssh.SSHKeygenResponse

Generate an SSH keypair.

Parameters

- **`filename`** – Output file name for the private key (the public key will be stored in <filename>.pub).
- **`type`** – Encryption algorithm, either "rsa" or "dsa" (default: "rsa").
- **`bits`** – Key length in bits (default: 4096).
- **`comment`** – Key comment (default: None).
- **`passphrase`** – Key passphrase (default: None).

Returns *`platypush.message.response.ssh.SSHKeygenResponse`*.

`ln`(*src: str*, *dest: str*, *keep_alive: bool = False*, ***kwargs*) → None

Create a symbolic link.

Parameters

- **`src`** – Source path.
- **`dest`** – Destination path.
- **`keep_alive`** – Keep the connection active after running the command (default: False).
- **`kwargs`** – Arguments for *`platypush.plugins.ssh.SshPlugin.connect()`*.

`ls`(*path: str = '.'*, *attrs: bool = False*, *keep_alive: bool = False*, ***kwargs*) → Union[List[str], Dict[str, Any]]

Return the list of files in a path on a remote server.

Parameters

- **`path`** – Remote path (default: current directory).
- **`keep_alive`** – Keep the connection active after running the command (default: False).
- **`attrs`** – Set to True if you want to get the full information of each file (default: False).

- **kwargs** – Arguments for *platypush.plugins.ssh.SshPlugin.connect()*.

Returns A list of filenames if `attrs=False`, otherwise a dictionary `filename -> {attributes` if `attrs=True`.

mkdir (*path: str*, *mode: int = 511*, *keep_alive: bool = False*, ***kwargs*) → None
Create a directory.

Parameters

- **path** – Path to be created.
- **mode** – Access permissions (default: 0777).
- **keep_alive** – Keep the connection active after running the command (default: False).
- **kwargs** – Arguments for *platypush.plugins.ssh.SshPlugin.connect()*.

mv (*path: str*, *new_path: str*, *keep_alive: bool = False*, ***kwargs*) → None
Move/rename a file.

Parameters

- **path** – Remote path to move/rename.
- **new_path** – Destination path.
- **keep_alive** – Keep the connection active after running the command (default: False).
- **kwargs** – Arguments for *platypush.plugins.ssh.SshPlugin.connect()*.

put (*remote_path: str*, *local_path: str*, *recursive: bool = False*, *keep_alive: bool = False*, ***kwargs*) → None
Upload a file or folder to an SSH server.

Parameters

- **remote_path** – Remote path (file or directory).
- **local_path** – Local path (file or directory).
- **recursive** – Set to True if you want to recursively upload folders (default: False).
- **keep_alive** – Keep the connection active after running the command (default: False).
- **kwargs** – Arguments for *platypush.plugins.ssh.SshPlugin.connect()*.

rm (*path: str*, *keep_alive: bool = False*, ***kwargs*) → None
Remove a file from the server.

Parameters

- **path** – Remote path to remove.
- **keep_alive** – Keep the connection active after running the command (default: False).
- **kwargs** – Arguments for *platypush.plugins.ssh.SshPlugin.connect()*.

rmdir(*path: str*, *keep_alive: bool = False*, ***kwargs*) → None
Remove a directory.

Parameters

- **path** – Path to be removed.
- **keep_alive** – Keep the connection active after running the command (default: False).
- **kwargs** – Arguments for `platypush.plugins.ssh.SshPlugin.connect()`.

start_forward_tunnel(*local_port: int*, *remote_host: str*, *remote_port: int*, *bind_addr: Optional[str] = ''*, ***kwargs*)
Start an SSH forward tunnel, tunnelling <local_port> to <remote_host>:<remote_port>.

Parameters

- **local_port** – Local port.
- **remote_host** – Remote host.
- **remote_port** – Remote port.
- **bind_addr** – If set, the *local_port* will be bound to this address/subnet (default: '', or 0.0.0.0: any).
- **kwargs** – Arguments for `platypush.plugins.ssh.SshPlugin.connect()`.

start_reverse_tunnel(*server_port: int*, *remote_host: str*, *remote_port: int*, *bind_addr: Optional[str] = ''*, ***kwargs*)
Start an SSH reversed tunnel. <server_port> on the SSH server is forwarded across an SSH session back to the local machine, and out to a <remote_host>:<remote_port> reachable from this network.

Parameters

- **server_port** – Server port.
- **remote_host** – Remote host.
- **remote_port** – Remote port.
- **bind_addr** – If set, the *server_port* will be bound to this address/subnet (default: '', or 0.0.0.0: any).
- **kwargs** – Arguments for `platypush.plugins.ssh.SshPlugin.connect()`.

stop_forward_tunnel(*local_port: int*, *remote_host: str*, *remote_port: int*)
Stop an active SSH forward tunnel.

Parameters

- **local_port** – Local port.
- **remote_host** – Remote host.
- **remote_port** – Remote port.

stop_reverse_tunnel(*server_port: int*, *remote_host: str*, *remote_port: int*)
Stop an active SSH reversed tunnel.

Parameters

- **server_port** – Server port.

- **remote_host** – Remote host.
- **remote_port** – Remote port.

2.109 platypush.plugins.stt

class platypush.plugins.stt.**SttPlugin**(*input_device: Union[str, int, None] = None, hotword: Optional[str] = None, hotwords: Optional[List[str]] = None, conversation_timeout: Optional[float] = 10.0, block_duration: float = 1.0*)

Abstract class for speech-to-text plugins.

Triggers:

- *platypush.message.event.stt.SpeechStartedEvent* when speech starts being detected.
- *platypush.message.event.stt.SpeechDetectedEvent* when speech is detected.
- *platypush.message.event.stt.SpeechDetectionStartedEvent* when speech detection starts.
- *platypush.message.event.stt.SpeechDetectionStoppedEvent* when speech detection stops.
- *platypush.message.event.stt.HotwordDetectedEvent* when a user-defined hotword is detected.
- *platypush.message.event.stt.ConversationDetectedEvent* when speech is detected after a hotword.

__init__(*input_device: Union[str, int, None] = None, hotword: Optional[str] = None, hotwords: Optional[List[str]] = None, conversation_timeout: Optional[float] = 10.0, block_duration: float = 1.0*)

Parameters

- **input_device** – PortAudio device index or name that will be used for recording speech (default: default system audio input device).
- **hotword** – When this word is detected, the plugin will trigger a *platypush.message.event.stt.HotwordDetectedEvent* instead of a *platypush.message.event.stt.SpeechDetectedEvent* event. You can use these events for hooking other assistants.
- **hotwords** – Use a list of hotwords instead of a single one.
- **conversation_timeout** – If `hotword` or `hotwords` are set and `conversation_timeout` is set, the next speech detected event will trigger a *platypush.message.event.stt.ConversationDetectedEvent* instead of a *platypush.message.event.stt.SpeechDetectedEvent* event. You can hook custom hooks here to run any logic depending on the detected speech - it can emulate a kind of "OK, Google. Turn on the lights" interaction without using an external assistant (default: 10 seconds).
- **block_duration** – Duration of the acquired audio blocks (default: 1 second).

before_recording() → None

Method called when the `recording_thread` starts. Put here any logic that you may want to run before the recording thread starts.

static convert_frames(*frames: bytes*) → bytes
Conversion method for raw audio frames. It just returns the input frames as bytes. Override it if required by your logic.

Parameters **frames** – Input audio frames, as bytes.

Returns The audio frames as passed on the input. Override if required.

detect(*audio_file: str*) → platypush.message.response.stt.SpeechDetectedResponse
Perform speech-to-text analysis on an audio file. Must be implemented by the derived classes.

Parameters **audio_file** – Path to the audio file.

detect_speech(*frames*) → str
Method called within the `detection_thread` when new audio frames have been captured. Must be implemented by the derived classes.

Parameters **frames** – Audio frames, as returned by `convert_frames`.

Returns Detected text, as a string. Returns an empty string if no text has been detected.

detection_thread() → None
This thread reads frames from `_audio_queue`, performs the speech-to-text detection and calls

on_detection_ended() → None
Method called when the `detection_thread` stops. Clean up your context variables and models here.

on_detection_started() → None
Method called when the `detection_thread` starts. Initialize your context variables and models here if required.

on_recording_ended() → None
Method called when the `recording_thread` stops. Put here any logic that you want to run after the audio device is closed.

on_recording_started() → None
Method called after the `recording_thread` opens the audio device. Put here any logic that you may want to run after the recording starts.

on_speech_detected(*speech: str*) → None
Hook called when speech is detected. Triggers the right event depending on the current context.

Parameters **speech** – Detected speech.

recording_thread(*block_duration: Optional[float] = None*, *block_size: Optional[int] = None*, *input_device: Optional[str] = None*) → None
Recording thread. It reads raw frames from the audio device and dispatches them to `detection_thread`.

Parameters

- **block_duration** – Audio blocks duration. Specify either `block_duration` or `block_size`.
- **block_size** – Size of the audio blocks. Specify either `block_duration` or `block_size`.
- **input_device** – Input device

start_detection(*input_device: Optional[str] = None*, *seconds: Optional[float] = None*, *block_duration: Optional[float] = None*) → None
Start the speech detection engine.

Parameters

- **input_device** – Audio input device name/index override
- **seconds** – If set, then the detection engine will stop after this many seconds, otherwise it'll start running until `stop_detection` is called or application stop.
- **block_duration** – `block_duration` override.

stop_detection() → None
Stop the speech detection engine.

2.110 platypush.plugins.stt.deepspeech

class platypush.plugins.stt.deepspeech.**SttDeepspeechPlugin**(*model_file: str, lm_file: str, trie_file: str, lm_alpha: float = 0.75, lm_beta: float = 1.85, beam_width: int = 500, *args, **kwargs*)

This plugin performs speech-to-text and speech detection using the Mozilla DeepSpeech engine.

Requires:

- **deepspeech** (`pip install 'deepspeech>=0.6.0'`)
- **numpy** (`pip install numpy`)
- **sounddevice** (`pip install sounddevice`)

__init__(*model_file: str, lm_file: str, trie_file: str, lm_alpha: float = 0.75, lm_beta: float = 1.85, beam_width: int = 500, *args, **kwargs*)

In order to run the speech-to-text engine you'll need to download the right model files for the Deepspeech engine that you have installed:

```
# Create the working folder for the models
export MODELS_DIR=~/models
mkdir -p $MODELS_DIR
cd $MODELS_DIR

# Download and extract the model files for your version of Deepspeech. This␣
↪may take a while.
export DEEPSPEECH_VERSION=0.6.1
wget https://github.com/mozilla/DeepSpeech/releases/download/v$DEEPSPEECH_
↪VERSION/deepspeech-$DEEPSPEECH_VERSION-models.tar.gz
tar -xvzf deepspeech-$DEEPSPEECH_VERSION-models.tar.gz
x deepspeech-0.6.1-models/
x deepspeech-0.6.1-models/lm.binary
x deepspeech-0.6.1-models/output_graph.pbmm
x deepspeech-0.6.1-models/output_graph.pb
x deepspeech-0.6.1-models/trie
x deepspeech-0.6.1-models/output_graph.tflite
```

Parameters

- **model_file** – Path to the model file (usually named `output_graph.pb` or `output_graph.pbmm`). Note that `.pbmm` usually perform better and are smaller.
- **lm_file** – Path to the language model binary file (usually named `lm.binary`).

- **trie_file** – The path to the trie file build from the same vocabulary as the language model binary (usually named `trie`).
- **lm_alpha** – The alpha hyperparameter of the CTC decoder - Language Model weight. See <https://github.com/mozilla/DeepSpeech/releases/tag/v0.6.0>.
- **lm_beta** – The beta hyperparameter of the CTC decoder - Word Insertion weight. See <https://github.com/mozilla/DeepSpeech/releases/tag/v0.6.0>.
- **beam_width** – Decoder beam width (see beam scoring in KenLM language model).
- **input_device** – PortAudio device index or name that will be used for recording speech (default: default system audio input device).
- **hotword** – When this word is detected, the plugin will trigger a *`platypush.message.event.stt.HotwordDetectedEvent`* instead of a *`platypush.message.event.stt.SpeechDetectedEvent`* event. You can use these events for hooking other assistants.
- **hotwords** – Use a list of hotwords instead of a single one.
- **conversation_timeout** – If `hotword` or `hotwords` are set and `conversation_timeout` is set, the next speech detected event will trigger a *`platypush.message.event.stt.ConversationDetectedEvent`* instead of a *`platypush.message.event.stt.SpeechDetectedEvent`* event. You can hook custom hooks here to run any logic depending on the detected speech - it can emulate a kind of "OK, Google. Turn on the lights" interaction without using an external assistant.
- **block_duration** – Duration of the acquired audio blocks (default: 1 second).

static convert_frames(*frames: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fc391a75850>, bytes]*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391a75450>
Conversion method for raw audio frames. It just returns the input frames as bytes. Override it if required by your logic.

Parameters **frames** – Input audio frames, as bytes.

Returns The audio frames as passed on the input. Override if required.

detect(*audio_file: str*) → platypush.message.response.stt.SpeechDetectedResponse
Perform speech-to-text analysis on an audio file.

Parameters **audio_file** – Path to the audio file.

detect_speech(*frames*) → str
Method called within the `detection_thread` when new audio frames have been captured. Must be implemented by the derived classes.

Parameters **frames** – Audio frames, as returned by `convert_frames`.

Returns Detected text, as a string. Returns an empty string if no text has been detected.

on_detection_ended()
Method called when the `detection_thread` stops. Clean up your context variables and models here.

on_detection_started()
Method called when the `detection_thread` starts. Initialize your context variables and models here if required.

on_speech_detected(*speech: str*) → None
Hook called when speech is detected. Triggers the right event depending on the current context.

Parameters **speech** – Detected speech.

2.111 platypush.plugins.stt.picovoice.hotword

class platypush.plugins.stt.picovoice.hotword.**SttPicovoiceHotwordPlugin**(*library_path: Optional[str] = None, model_file_path: Optional[str] = None, keyword_file_paths: Optional[List[str]] = None, sensitivity: float = 0.5, sensitivities: Optional[List[float]] = None, *args, **kwargs*)

This plugin performs hotword detection using PicoVoice.

Requires:

- **pvporcupine** (`pip install pvporcupine`) for hotword detection.

__init__(*library_path: Optional[str] = None, model_file_path: Optional[str] = None, keyword_file_paths: Optional[List[str]] = None, sensitivity: float = 0.5, sensitivities: Optional[List[float]] = None, *args, **kwargs*)

Parameters

- **input_device** – PortAudio device index or name that will be used for recording speech (default: default system audio input device).
- **hotword** – When this word is detected, the plugin will trigger a *platypush.message.event.stt.HotwordDetectedEvent* instead of a *platypush.*

message.event.stt.SpeechDetectedEvent event. You can use these events for hooking other assistants.

- **hotwords** – Use a list of hotwords instead of a single one.
- **conversation_timeout** – If `hotword` or `hotwords` are set and `conversation_timeout` is set, the next speech detected event will trigger a *platypush.message.event.stt.ConversationDetectedEvent* instead of a *platypush.message.event.stt.SpeechDetectedEvent* event. You can hook custom hooks here to run any logic depending on the detected speech - it can emulate a kind of "OK, Google. Turn on the lights" interaction without using an external assistant (default: 10 seconds).
- **block_duration** – Duration of the acquired audio blocks (default: 1 second).

convert_frames(*frames: bytes*) → tuple
Conversion method for raw audio frames. It just returns the input frames as bytes. Override it if required by your logic.

Parameters **frames** – Input audio frames, as bytes.

Returns The audio frames as passed on the input. Override if required.

detect(*audio_file: str*) → platypush.message.response.stt.SpeechDetectedResponse
Perform speech-to-text analysis on an audio file.

Parameters **audio_file** – Path to the audio file.

detect_speech(*frames: tuple*) → str
Method called within the `detection_thread` when new audio frames have been captured. Must be implemented by the derived classes.

Parameters **frames** – Audio frames, as returned by `convert_frames`.

Returns Detected text, as a string. Returns an empty string if no text has been detected.

on_detection_ended() → None
Method called when the `detection_thread` stops. Clean up your context variables and models here.

recording_thread(*input_device: Optional[str] = None*, **args*, ***kwargs*) → None
Recording thread. It reads raw frames from the audio device and dispatches them to `detection_thread`.

Parameters

- **block_duration** – Audio blocks duration. Specify either `block_duration` or `block_size`.
- **block_size** – Size of the audio blocks. Specify either `block_duration` or `block_size`.
- **input_device** – Input device

start_detection(**args*, ***kwargs*) → None
Start the speech detection engine.

Parameters

- **input_device** – Audio input device name/index override
- **seconds** – If set, then the detection engine will stop after this many seconds, otherwise it'll start running until `stop_detection` is called or application stop.
- **block_duration** – `block_duration` override.

2.112 `platypush.plugins.stt.picovoice.speech`

class `platypush.plugins.stt.picovoice.speech.`**`SttPicovoiceSpeechPlugin`**(*library_path: Optional[str] = None, acoustic_model_path: Optional[str] = None, language_model_path: Optional[str] = None, license_path: Optional[str] = None, end_of_speech_timeout: int = 1, *args, **kwargs*)

This plugin performs speech detection using PicoVoice. NOTE: The PicoVoice product used for real-time speech-to-text (Cheetah) can be used freely for personal applications on x86_64 Linux. Other architectures and operating systems require a commercial license. You can ask for a license here.

Requires:

- **cheetah** (`pip install git+https://github.com/BlackLight/cheetah`)

`__init__`(*library_path: Optional[str] = None, acoustic_model_path: Optional[str] = None, language_model_path: Optional[str] = None, license_path: Optional[str] = None, end_of_speech_timeout: int = 1, *args, **kwargs*)

Parameters

- **`library_path`** – Path to the Cheetah binary library for your OS (default: `CHEETAH_INSTALL_DIR/lib/OS/ARCH/libpv_cheetah.EXT`).
- **`acoustic_model_path`** – Path to the acoustic speech model (default: `CHEETAH_INSTALL_DIR/lib/common/acoustic_model.pv`).
- **`language_model_path`** – Path to the language model (default: `CHEETAH_INSTALL_DIR/lib/common/language_model.pv`).
- **`license_path`** – Path to your PicoVoice license (default: `CHEETAH_INSTALL_DIR/resources/license/cheetah_eval_linux_public.lic`).

- **end_of_speech_timeout** – Number of seconds of silence during speech recognition before considering a phrase over (default: 1).

convert_frames (*frames: bytes*) → tuple
Conversion method for raw audio frames. It just returns the input frames as bytes. Override it if required by your logic.

Parameters **frames** – Input audio frames, as bytes.

Returns The audio frames as passed on the input. Override if required.

detect (*audio_file: str*) → platypush.message.response.stt.SpeechDetectedResponse
Perform speech-to-text analysis on an audio file.

Parameters **audio_file** – Path to the audio file.

detect_speech (*frames: tuple*) → str
Method called within the `detection_thread` when new audio frames have been captured. Must be implemented by the derived classes.

Parameters **frames** – Audio frames, as returned by `convert_frames`.

Returns Detected text, as a string. Returns an empty string if no text has been detected.

on_detection_ended () → None
Method called when the `detection_thread` stops. Clean up your context variables and models here.

recording_thread (*input_device: Optional[str] = None*, **args*, ***kwargs*) → None
Recording thread. It reads raw frames from the audio device and dispatches them to `detection_thread`.

Parameters

- **block_duration** – Audio blocks duration. Specify either `block_duration` or `block_size`.
- **block_size** – Size of the audio blocks. Specify either `block_duration` or `block_size`.
- **input_device** – Input device

start_detection (**args*, ***kwargs*) → None
Start the speech detection engine.

Parameters

- **input_device** – Audio input device name/index override
- **seconds** – If set, then the detection engine will stop after this many seconds, otherwise it'll start running until `stop_detection` is called or application stop.
- **block_duration** – `block_duration` override.

2.113 platypush.plugins.switch

class platypush.plugins.switch.**SwitchPlugin** (***kwargs*)
Abstract class for interacting with switch devices

__init__ (***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

off (*device*, **args*, ***kwargs*)
Turn the device off

on (*device, *args, **kwargs*)
Turn the device on

status (*device=None, *args, **kwargs*)
Status function - if not overridden it calls `switch_status()`. You may want to override it if your plugin does not handle only switches.

switch_status (*device=None*)
Get the status of a specified device or of all the configured devices (default)

switches
This property must be implemented by the derived classes and must return a dictionary in the following format:

```
[
    {
        "name": "switch_1",
        "on": true
    },
    {
        "name": "switch_2",
        "on": false
    },
]
```

`name` and `on` are the minimum set of attributes that should be returned for a switch, but more attributes can also be added.

toggle (*device, *args, **kwargs*)
Toggle the device status (on/off)

2.114 platypush.plugins.switch.switchbot

class platypush.plugins.switch.switchbot.**SwitchSwitchbotPlugin** (*interface=None, connect_timeout=None, scan_timeout=2, devices=None, **kwargs*)

Plugin to interact with a Switchbot (https://www.switch-bot.com/) device and programmatically control buttons.

See `platypush.plugins.bluetooth.ble.BluetoothBlePlugin` for how to enable BLE permissions for the platypush user (a simple solution may be to run it as root, but that's usually NOT a good idea).

Requires:

- **pybluez** (`pip install pybluez`)
- **gattlib** (`pip install gattlib`)
- **libboost** (on Debian `` `apt-get install libboost-python-dev libboost-thread-dev ``)

class Command
Base64 encoded commands

__init__ (*interface=None, connect_timeout=None, scan_timeout=2, devices=None, **kwargs*)

Parameters

- **`interface`** (*str*) – Bluetooth interface to use (e.g. hci0) default: first available one
- **`connect_timeout`** (*float*) – Timeout for the connection to the Switchbot device - default: None
- **`scan_timeout`** (*float*) – Timeout for the scan operations
- **`devices`** (*dict*) – Devices to control, as a MAC address -> name map

`off`(*device*, ***kwargs*)
Send a press-off button command to a device

Parameters **`device`** (*str*) – Device name or address

`on`(*device*, ***kwargs*)
Send a press-on button command to a device

Parameters **`device`** (*str*) – Device name or address

`press`(*device*)
Send a press button command to a device

Parameters **`device`** (*str*) – Device name or address

`scan`(*interface: str = None*, *duration: int = 10*) → platypush.message.response.bluetooth.BluetoothScanResponse
Scan for available Switchbot devices nearby.

Parameters

- **`interface`** – Bluetooth interface to scan (default: default configured interface)
- **`duration`** – Scan duration in seconds

`switches`
This property must be implemented by the derived classes and must return a dictionary in the following format:

```
[
    {
        "name": "switch_1",
        "on": true
    },
    {
        "name": "switch_2",
        "on": false
    },
]
```

`name` and `on` are the minimum set of attributes that should be returned for a switch, but more attributes can also be added.

`toggle`(*device*, ***kwargs*)
Toggle the device status (on/off)

2.115 `platypush.plugins.switch.tplink`

class `platypush.plugins.switch.tplink.`**`SwitchTplinkPlugin`**(*plugs: Union[Dict[str, str], List[str]] = None, bulbs: Union[Dict[str, str], List[str]] = None, strips: Union[Dict[str, str], List[str]] = None, **kwargs*)

Plugin to interact with TP-Link smart switches/plugs like the HS100 (https://www.tp-link.com/us/products/details/cat-5516_HS100.html).

Requires:

- **pyHS100** (`pip install pyHS100`)

`__init__`(*plugs: Union[Dict[str, str], List[str]] = None, bulbs: Union[Dict[str, str], List[str]] = None, strips: Union[Dict[str, str], List[str]] = None, **kwargs*)

Parameters

- **`plugs`** – Optional list of IP addresses or name->address mapping if you have a static list of TpLink plugs and you want to save on the scan time.
- **`bulbs`** – Optional list of IP addresses or name->address mapping if you have a static list of TpLink bulbs and you want to save on the scan time.
- **`strips`** – Optional list of IP addresses or name->address mapping if you have a static list of TpLink strips and you want to save on the scan time.

`off`(*device, **kwargs*)

Turn off a device

Parameters **`device`** (`str`) – Device IP, hostname or alias

`on`(*device, **kwargs*)

Turn on a device

Parameters **`device`** (`str`) – Device IP, hostname or alias

`switches`

This property must be implemented by the derived classes and must return a dictionary in the following format:

```
[
    {
        "name": "switch_1",
        "on": true
    },
    {
        "name": "switch_2",
        "on": false
    },
]
```

`name` and `on` are the minimum set of attributes that should be returned for a switch, but more attributes can also be added.

`toggle`(*device, **kwargs*)

Toggle the state of a device (on/off)

Parameters **`device`** (`str`) – Device IP, hostname or alias

2.116 `platypush.plugins.switch.wemo`

class `platypush.plugins.switch.wemo.`**`SwitchWemoPlugin`**(*devices=None*, *netmask: str = None*, *port: int = 49153*, ***kwargs*)

Plugin to control a Belkin WeMo smart switches (https://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/)

Requires:

- **requests** (`pip install requests`)

`__init__`(*devices=None*, *netmask: str = None*, *port: int = 49153*, ***kwargs*)

Parameters

- **`devices`** (*`list or dict`*) – List of IP addresses or name->address map containing the WeMo Switch devices to control. This plugin previously used ouimeaux for auto-discovery but it's been dropped because 1. too slow 2. too heavy 3. auto-discovery failed too often.
- **`netmask`** – Alternatively to a list of static IP->name pairs, you can specify the network mask where the devices should be scanned (e.g. '192.168.1.0/24')
- **`port`** – Port where the WeMo devices are expected to expose the RPC/XML over HTTP service (default: 49153)

`get_name`(*device: str*)

Get the friendly name of a device

Parameters **`device`** – Device name or address

`get_state`(*device: str*)

Get the on state of a device (True/False)

Parameters **`device`** – Device name or address

`off`(*device: str*, ***kwargs*)

Turn a switch off

Parameters **`device`** – Device name or address

`on`(*device: str*, ***kwargs*)

Turn a switch on

Parameters **`device`** – Device name or address

`status`(*device: str = None*, **args*, ***kwargs*)

Status function - if not overridden it calls *`switch_status()`*. You may want to override it if your plugin does not handle only switches.

`switches`

Get the list of available devices :returns: The list of devices.

```
[
    {
        "ip": "192.168.1.123",
        "name": "Switch 1",
        "on": true
    },
    {
        "ip": "192.168.1.124",
```

(continues on next page)

(continued from previous page)

```
        "name": "Switch 2",
        "on": false
    }
]
```

toggle (*device: str*, **args*, ***kwargs*)
Toggle a device on/off state

Parameters **device** – Device name or address

2.117 platypush.plugins.system

class platypush.plugins.system.**SystemPlugin** (***kwargs*)
Plugin to get system info.

Requires:

- **py-cpuinfo** (`pip install py-cpuinfo`) for CPU model and info.
- **psutil** (`pip install psutil`) for CPU load and stats.

connected_users () → platypush.message.response.system.ConnectedUserResponseList
Get the list of connected users. :return: List of `platypush.message.response.system.ConnectUserResponse`.

cpu_frequency (*per_cpu: bool = False*) → Union[platypush.message.response.system.CpuFrequencyResponse, platypush.message.response.system.CpuResponseList]
Get CPU stats.

Parameters **per_cpu** – Get per-CPU stats (default: False).

Returns `platypush.message.response.system.CpuFrequencyResponse`

cpu_info () → platypush.message.response.system.CpuInfoResponse
Get CPU info. :return: `platypush.message.response.system.CpuInfoResponse`

cpu_percent (*per_cpu: bool = False*, *interval: Optional[float] = None*) → Union[float, List[float]]
Get the CPU load percentage.

Parameters

- **per_cpu** – Get per-CPU stats (default: False).
- **interval** – When *interval* is 0.0 or None compares system CPU times elapsed since last call or module import, returning immediately (non blocking). That means the first time this is called it will return a meaningless 0.0 value which you should ignore. In this case is recommended for accuracy that this function be called with at least 0.1 seconds between calls.

Returns float if `per_cpu=False`, `list[float]` otherwise.

cpu_stats () → platypush.message.response.system.CpuStatsResponse
Get CPU stats. :return: `platypush.message.response.system.CpuStatsResponse`

cpu_times (*per_cpu=False*, *percent=False*) → Union[platypush.message.response.system.CpuTimesResponse, platypush.message.response.system.CpuResponseList]
Get the CPU times stats.

Parameters

- **per_cpu** – Get per-CPU stats (default: False).

- **percent** – Get the stats in percentage (default: False).

Returns *platypush.message.response.system.CpuTimesResponse*

disk_io_counters(*disk: Optional[str] = None, per_disk: bool = False*) → Union[platypush.message.response.system.DiskIoCountersResponse, platypush.message.response.system.DiskResponseList]

Get the I/O counter stats for the mounted disks.

Parameters

- **disk** – Select the stats for a specific disk (e.g. 'sda1'). Default: get stats for all mounted disks.
- **per_disk** – Return the stats per disk (default: False).

Returns *platypush.message.response.system.DiskIoCountersResponse* or list of *platypush.message.response.system.DiskIoCountersResponse*.

disk_partitions() → platypush.message.response.system.DiskResponseList

Get the list of partitions mounted on the system. :return: list of *platypush.message.response.system.DiskPartitionResponse*

disk_usage(*path: Optional[str] = None*) → Union[platypush.message.response.system.DiskUsageResponse, platypush.message.response.system.DiskResponseList]

Get the usage of a mounted disk.

Parameters **path** – Path where the device is mounted (default: get stats for all mounted devices).

Returns *platypush.message.response.system.DiskUsageResponse* or list of *platypush.message.response.system.DiskUsageResponse*.

kill(*pid: int*)

Kill a process. :param pid: Process PID.

load_avg() → List[float]

Get the average load as a vector that represents the load within the last 1, 5 and 15 minutes.

mem_swap() → platypush.message.response.system.SwapMemoryUsageResponse

Get the current virtual memory usage stats. :return: list of *platypush.message.response.system.SwapMemoryUsageResponse*

mem_virtual() → platypush.message.response.system.VirtualMemoryUsageResponse

Get the current virtual memory usage stats. :return: list of *platypush.message.response.system.VirtualMemoryUsageResponse*

net_addresses(*nic: Optional[str] = None*) → Union[platypush.message.response.system.NetworkAddressResponse, platypush.message.response.system.NetworkResponseList]

Get address info associated to the network interfaces.

Parameters **nic** – Select the stats for a specific network device (e.g. 'eth0'). Default: get stats for all NICs.

Returns *platypush.message.response.system.NetworkAddressResponse* or list of *platypush.message.response.system.NetworkAddressResponse*.

net_connections(*type: Optional[str] = None*) → Union[platypush.message.response.system.NetworkConnectionResponse platypush.message.response.system.NetworkResponseList]

Get the list of active network connections. On macOS this function requires root privileges.

Parameters **`type`** – Connection type to filter. Supported types:

Kind Value	Connections using
inet inet4 inet6 tcp tcp4 tcp6 udp udp4 udp6 unix all	IPv4 and IPv6 IPv4 IPv6 TCP TCP over IPv4 TCP over IPv6 UDP UDP over IPv4 UDP over IPv6 UNIX socket (both UDP and TCP protocols) the sum of all the possible families and protocols

Returns List of *`platypush.message.response.system.NetworkConnectionResponse`*.

`net_io_counters`(*nic: Optional[str] = None*, *per_nic: bool = False*) → Union[platypush.message.response.system.NetworkIoCountersResponse, platypush.message.response.system.NetworkResponseList]

Get the I/O counters stats for the network interfaces.

Parameters

- **`nic`** – Select the stats for a specific network device (e.g. 'eth0'). Default: get stats for all NICs.
- **`per_nic`** – Return the stats broken down per interface (default: False).

Returns `platypush.message.response.system.NetIoCountersResponse` or list of `platypush.message.response.system.NetIoCountersResponse`.

`net_stats`(*nic: Optional[str] = None*) → Union[platypush.message.response.system.NetworkInterfaceStatsResponse, platypush.message.response.system.NetworkResponseList]

Get stats about the network interfaces.

Parameters **`nic`** – Select the stats for a specific network device (e.g. 'eth0'). Default: get stats for all NICs.

Returns *`platypush.message.response.system.NetworkInterfaceStatsResponse`* or list of *`platypush.message.response.system.NetworkInterfaceStatsResponse`*.

`pid_exists`(*pid: int*) → bool

Parameters **`pid`** – Process PID.

Returns `True` if the process exists, `False` otherwise.

`processes`(*filter: Optional[str] = ''*) → platypush.message.response.system.ProcessResponseList

Get the list of running processes.

Parameters **`filter`** – Filter the list by name.

Returns List of *`platypush.message.response.system.ProcessResponse`*.

`resume`(*pid: int*)

Resume a process. :param pid: Process PID.

`sensors_battery`() → platypush.message.response.system.SensorBatteryResponse

Get stats from the battery sensor. :return: List of *`platypush.message.response.system.SensorFanResponse`*.

`sensors_fan`(*sensor: Optional[str] = None*) → platypush.message.response.system.SensorResponseList

Get stats from the fan sensors.

Parameters **`sensor`** – Select the sensor name.

Returns List of *platypush.message.response.system.SensorFanResponse*.

sensors_temperature(*sensor: Optional[str] = None*, *fahrenheit: bool = False*) → Union[platypush.message.response.system.SensorTemperatureResponse, List[platypush.message.response.system.SensorTemperatureResponse], Dict[str, Union[platypush.message.response.system.SensorTemperatureResponse, List[platypush.message.response.system.SensorTemperatureResponse]]]]

Get stats from the temperature sensors.

Parameters

- **sensor** – Select the sensor name.
- **fahrenheit** – Return the temperature in Fahrenheit (default: Celsius).

suspend(*pid: int*)

Suspend a process. :param pid: Process PID.

terminate(*pid: int*)

Terminate a process. :param pid: Process PID.

wait(*pid: int*, *timeout: int = None*)

Wait for a process to terminate.

Parameters

- **pid** – Process PID.
- **timeout** – Timeout in seconds (default: `None`).

2.118 platypush.plugins.tcp

class platypush.plugins.tcp.**TcpPlugin**(***kwargs*)

Plugin for raw TCP communications.

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

close(*host: str*, *port: int*)

Close an active TCP connection.

Parameters

- **host** – Host IP/name.
- **port** – TCP port.

connect(*host: str*, *port: int*, *timeout: Optional[float] = None*)

Open a TCP connection.

Parameters

- **host** – Host IP/name.
- **port** – TCP port.
- **timeout** – Connection timeout in seconds (default: None).

recv(*length: int*, *host: str*, *port: int*, *binary: bool = False*, *timeout: Optional[float] = None*) → str

Receive data from a TCP connection. If the connection isn't active it will be created.

Parameters

- **length** – Maximum number of bytes to be received.

- **host** – Host IP/name.
- **port** – TCP port.
- **binary** – If set to True then the output will be base64-encoded, otherwise decoded as string.
- **timeout** – Connection timeout in seconds (default: None).

send(*data: Union[bytes, str], host: str, port: int, binary: bool = False, timeout: Optional[float] = None, recv_response: bool = False, **recv_opts*)

Send data over a TCP connection. If the connection isn't active it will be created.

Parameters

- **data** – Data to be sent, as bytes or string.
- **host** – Host IP/name.
- **port** – TCP port.
- **binary** – If set to True and `data` is a string then will be treated as base64-encoded binary input.
- **timeout** – Connection timeout in seconds (default: None).
- **recv_response** – If True then the action will wait for a response from the server before closing the connection. Note that `recv_opts` must be specified in this case - at least `length`.

2.119 platypush.plugins.tensorflow

class platypush.plugins.tensorflow.**TensorflowPlugin**(*workdir: Optional[str] = None, **kwargs*)

This plugin can be used to create, train, load and make predictions with TensorFlow-compatible machine learning models.

Triggers:

- *platypush.message.event.tensorflow.TensorflowEpochStartedEvent* when a Tensorflow model training/evaluation epoch begins.
- *platypush.message.event.tensorflow.TensorflowEpochEndedEvent* when a Tensorflow model training/evaluation epoch ends.
- *platypush.message.event.tensorflow.TensorflowBatchStartedEvent* when a Tensorflow model training/evaluation batch starts being processed.
- *platypush.message.event.tensorflow.TensorflowBatchEndedEvent* when a the processing of a Tensorflow model training/evaluation batch ends.
- *platypush.message.event.tensorflow.TensorflowTrainStartedEvent* when a Tensorflow model starts being trained.
- *platypush.message.event.tensorflow.TensorflowTrainEndedEvent* when the training phase of a Tensorflow model ends.

Requires:

- **numpy** (`pip install numpy`)
- **pandas** (`pip install pandas`) (optional, for CSV parsing)
- **tensorflow** (`pip install 'tensorflow>=2.0'`)

- **keras** (`pip install keras`)

`__init__`(*workdir: Optional[str] = None*, ***kwargs*)

Parameters **`workdir`** – Working directory for TensorFlow, where models will be stored and looked up by default (default: PLATYPUSH_WORKDIR/tensorflow).

`create_network`(*name: str*, *layers: List[Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fc391cafcd0>, Dict[str, Any]]]*, *input_names: Optional[List[str]] = None*, *output_names: Optional[List[str]] = None*, *optimizer: Optional[str] = 'rmsprop'*, *loss: Union[str, List[str], Dict[str, str], None] = None*, *metrics: Union[str, List[Union[str, List[str]]], Dict[str, Union[str, List[str]]], None] = None*, *loss_weights: Union[List[float], Dict[str, float], None] = None*, *sample_weight_mode: Union[str, List[str], Dict[str, str], None] = None*, *weighted_metrics: Optional[List[str]] = None*, *target_tensors=None*, ***kwargs*) → Dict[str, Any]

Create a neural network TensorFlow Keras model.

Parameters

- **`name`** – Name of the model.
- **`layers`** – List of layers. Example:

```
[
  // Input flatten layer with 10 units
  {
    "type": "Flatten",
    "input_shape": [10, 10]
  },

  // Dense hidden layer with 500 units
  {
    "type": "Dense",
    "units": 500,
    "activation": "relu"
  },

  // Dense hidden layer with 100 units
  {
    "type": "Dense",
    "units": 100,
    "activation": "relu"
  },

  // Dense output layer with 2 units (labels) and ``softmax``␣
↪activation function
  {
    "type": "Dense",
    "units": 2,
    "activation": "softmax"
  }
]
```

- **`input_names`** – List of names for the input units (default: TensorFlow name auto-assign logic).
- **`output_names`** – List of labels for the output units (default: TensorFlow name auto-assign logic).
- **`optimizer`** – Optimizer, see <https://keras.io/optimizers/> (default: `rmsprop`).

- **loss** – Loss function, see <https://keras.io/losses/>. An objective function is any callable with the signature `scalar_loss = fn(y_true, y_pred)`. If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses (default: None).
- **metrics** – List of metrics to be evaluated by the model during training and testing. Typically you will use `metrics=['accuracy']`. To specify different metrics for different outputs of a multi-output model, you could also pass a dictionary, such as `metrics={'output_a': 'accuracy', 'output_b': ['accuracy', 'mse']}`. You can also pass a list `(len = len(outputs))` of lists of metrics such as `metrics=[['accuracy'], ['accuracy', 'mse']]` or `metrics=['accuracy', ['accuracy', 'mse']]`. Default: `['accuracy']`.
- **loss_weights** – Optional list or dictionary specifying scalar coefficients (Python floats) to weight the loss contributions of different model outputs. The loss value that will be minimized by the model will then be the *weighted sum* of all individual losses, weighted by the *loss_weights* coefficients. If a list, it is expected to have a 1:1 mapping to the model's outputs. If a tensor, it is expected to map output names (strings) to scalar coefficients.
- **sample_weight_mode** – If you need to do time-step-wise sample weighting (2D weights), set this to `"temporal"`. `None` defaults to sample-wise weights (1D). If the model has multiple outputs, you can use a different `sample_weight_mode` on each output by passing a dictionary or a list of modes.
- **weighted_metrics** – List of metrics to be evaluated and weighted by `sample_weight` or `class_weight` during training and testing.
- **target_tensors** – By default, Keras will create placeholders for the model's target, which will be fed with the target data during training. If instead you would like to use your own target tensors (in turn, Keras will not expect external numpy data for these targets at training time), you can specify them via the `target_tensors` argument. It can be a single tensor (for a single-output model), a list of tensors, or a dict mapping output names to target tensors.
- **kwargs** – Extra arguments to pass to `Model.compile()`.

Returns

The model configuration, as a dict. Example:

```
{
  "name": "test_model",
  "layers": [
    {
      "class_name": "Flatten",
      "config": {
        "name": "flatten",
        "trainable": true,
        "batch_input_shape": [
          null,
          10
        ],
        "dtype": "float32",
        "data_format": "channels_last"
      }
    },
```

(continues on next page)

(continued from previous page)

```
    {
      "class_name": "Dense",
      "config": {
        "name": "dense",
        "trainable": true,
        "dtype": "float32",
        "units": 100,
        "activation": "relu",
        "use_bias": true,
        "kernel_initializer": {
          "class_name": "GlorotUniform",
          "config": {
            "seed": null
          }
        },
        "bias_initializer": {
          "class_name": "Zeros",
          "config": {}
        },
        "kernel_regularizer": null,
        "bias_regularizer": null,
        "activity_regularizer": null,
        "kernel_constraint": null,
        "bias_constraint": null
      }
    },
    {
      "class_name": "Dense",
      "config": {
        "name": "dense_1",
        "trainable": true,
        "dtype": "float32",
        "units": 50,
        "activation": "relu",
        "use_bias": true,
        "kernel_initializer": {
          "class_name": "GlorotUniform",
          "config": {
            "seed": null
          }
        },
        "bias_initializer": {
          "class_name": "Zeros",
          "config": {}
        },
        "kernel_regularizer": null,
        "bias_regularizer": null,
        "activity_regularizer": null,
        "kernel_constraint": null,
        "bias_constraint": null
      }
    },
    {
      "class_name": "Dense",
      "config": {
        "name": "dense_2",
        "trainable": true,
```

(continues on next page)

(continued from previous page)

```
          "dtype": "float32",
          "units": 2,
          "activation": "softmax",
          "use_bias": true,
          "kernel_initializer": {
            "class_name": "GlorotUniform",
            "config": {
              "seed": null
            }
          },
          "bias_initializer": {
            "class_name": "Zeros",
            "config": {}
          },
          "kernel_regularizer": null,
          "bias_regularizer": null,
          "activity_regularizer": null,
          "kernel_constraint": null,
          "bias_constraint": null
        }
      }
    ]
}
```

create_regression(*name: str, units: int = 1, input_names: Optional[List[str]] = None, output_names: Optional[List[str]] = None, activation: str = 'linear', use_bias: bool = True, kernel_initializer: str = 'glorot_uniform', bias_initializer: str = 'zeros', kernel_regularizer: Optional[str] = None, bias_regularizer: Optional[str] = None, optimizer: Optional[str] = 'rmsprop', loss: Union[str, List[str], Dict[str, str], None] = 'mse', metrics: Union[str, List[Union[str, List[str]]], Dict[str, Union[str, List[str]]], None] = None, loss_weights: Union[List[float], Dict[str, float], None] = None, sample_weight_mode: Union[str, List[str], Dict[str, str], None] = None, weighted_metrics: Optional[List[str]] = None, target_tensors=None, **kwargs*) → Dict[str, Any]

Create a linear/logistic regression model.

Parameters

- **name** – Name of the model.
- **units** – Output dimension (default: 1).
- **input_names** – List of names for the input units (default: TensorFlow name auto-assign logic).
- **output_names** – List of labels for the output units (default: TensorFlow name auto-assign logic).
- **activation** – Activation function to be used (default: None).
- **use_bias** – Whether to calculate the bias/intercept for this model. If set to False, no bias/intercept will be used in calculations, e.g., the data is already centered (default: True).
- **kernel_initializer** – Initializer for the `kernel` weights matrices (default: `glorot_uniform`).
- **bias_initializer** – Initializer for the bias vector (default: `zeros`).
- **kernel_regularizer** – Regularizer for kernel vectors (default: None).

- **bias_regularizer** – Regularizer for bias vectors (default: None).
- **optimizer** – Optimizer, see <https://keras.io/optimizers/> (default: `rmsprop`).
- **loss** – Loss function, see <https://keras.io/losses/>. An objective function is any callable with the signature `scalar_loss = fn(y_true, y_pred)`. If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of losses. The loss value that will be minimized by the model will then be the sum of all individual losses (default: `mse`, mean squared error).
- **metrics** – List of metrics to be evaluated by the model during training and testing. Typically you will use `metrics=['accuracy']`. To specify different metrics for different outputs of a multi-output model, you could also pass a dictionary, such as `metrics={'output_a': 'accuracy', 'output_b': ['accuracy', 'mse']}`. You can also pass a list `(len = len(outputs))` of lists of metrics such as `metrics=[['accuracy'], ['accuracy', 'mse']]` or `metrics=['accuracy', ['accuracy', 'mse']]`. Default: `['mae', 'mse']`.
- **loss_weights** – Optional list or dictionary specifying scalar coefficients (Python floats) to weight the loss contributions of different model outputs. The loss value that will be minimized by the model will then be the *weighted sum* of all individual losses, weighted by the *loss_weights* coefficients. If a list, it is expected to have a 1:1 mapping to the model's outputs. If a tensor, it is expected to map output names (strings) to scalar coefficients.
- **sample_weight_mode** – If you need to do time-step-wise sample weighting (2D weights), set this to `"temporal"`. `None` defaults to sample-wise weights (1D). If the model has multiple outputs, you can use a different `sample_weight_mode` on each output by passing a dictionary or a list of modes.
- **weighted_metrics** – List of metrics to be evaluated and weighted by `sample_weight` or `class_weight` during training and testing.
- **target_tensors** – By default, Keras will create placeholders for the model's target, which will be fed with the target data during training. If instead you would like to use your own target tensors (in turn, Keras will not expect external numpy data for these targets at training time), you can specify them via the `target_tensors` argument. It can be a single tensor (for a single-output model), a list of tensors, or a dict mapping output names to target tensors.
- **kwargs** – Extra arguments to pass to `Model.compile()`.

Returns

Configuration of the model, as a dict. Example:

```
{
  "name": "test_regression_model",
  "trainable": true,
  "dtype": "float32",
  "units": 1,
  "activation": "linear",
  "use_bias": true,
  "kernel_initializer": {
    "class_name": "GlorotUniform",
    "config": {
      "seed": null
    }
```

(continues on next page)

(continued from previous page)

```
  },
  "bias_initializer": {
    "class_name": "Zeros",
    "config": {}
  },
  "kernel_regularizer": null,
  "bias_regularizer": null
}
```

evaluate(*model: str, inputs: Union[str, <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391a88650>, Iterable[T_co], Dict[str, Union[Iterable[T_co], <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391a88550>]]], outputs: Union[str, <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391a88c50>, Iterable[T_co], None] = None, batch_size: Optional[int] = None, verbose: int = 1, sample_weight: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fc391754110>, Iterable[T_co], None] = None, steps: Optional[int] = None, max_queue_size: int = 10, workers: int = 1, use_multiprocessing: bool = False*) → Union[Dict[str, float], List[float]]

Returns the loss value and metrics values for the model in test model.

Parameters

- **model** – Name of the model. It can be a folder name stored under `<workdir>/models`, or an absolute path to a model directory or file (Tensorflow directories, Protobuf models and HDF5 files are supported).
- **inputs** – Input data. It can be:
  - A numpy array (or array-like), or a list of arrays in case the model has multiple inputs.
  - A TensorFlow tensor, or a list of tensors in case the model has multiple inputs.
  - A dict mapping input names to the corresponding array/tensors, if the model has named inputs.
  - A `tf.data` dataset. Should return a tuple of either `(inputs, targets)` or `(inputs, targets, sample_weights)`.
  - A generator or `keras.utils.Sequence` returning `(inputs, targets)` or `(inputs, targets, sample weights)`.
  - A string that points to a file. Supported formats:
    - CSV with header (`.csv` extension‘‘)
    - Numpy raw or compressed files (`.npy` or `.npz` extension)
    - Image files
    - An HTTP URL pointing to one of the file types listed above
    - Directories with images. If `inputs` points to a directory of images then the following conventions are followed:
      - The folder must contain exactly as many subfolders as the output units of your model. If the model has `output_labels` then those subfolders should be named as the output labels. Each subfolder will contain training examples that match the associated label (e.g. `positive` will contain all the positive images and `negative` all the negative images).
      - `outputs` doesn’t have to be specified.

- **outputs** – Target data. Like the input data *x*, it can be a numpy array (or array-like) or TensorFlow tensor(s). It should be consistent with *x* (you cannot have Numpy inputs and tensor targets, or inversely). If *x* is a dataset, generator, or *keras.utils.Sequence* instance, *y* should not be specified (since targets will be obtained from *x*).
- **batch_size** – Number of samples per gradient update. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of symbolic tensors, datasets, generators, or `keras.utils.Sequence` instances (since they generate batches).
- **verbose** – Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch. Note that the progress bar is not particularly useful when logged to a file, so verbose=2 is recommended when not running interactively (eg, in a production environment).
- **sample_weight** – Optional iterable/numpy array of weights for the training samples, used for weighting the loss function (during training only). You can either pass a flat (1D) numpy array/iterable with the same length as the input samples (1:1 mapping between weights and samples), or in the case of temporal data, you can pass a 2D array with shape `(samples, sequence_length)`, to apply a different weight to every time step of every sample. In this case you should make sure to specify `sample_weight_mode="temporal"` in `compile()`. This argument is not supported when `x` is a dataset, generator, or `keras.utils.Sequence` instance, instead provide the sample_weights as the third element of `x`.
- **steps** – Total number of steps (batches of samples) before declaring the evaluation round finished. Ignored with the default value of `None`. If x is a `tf.data` dataset and `steps` is None, 'evaluate' will run until the dataset is exhausted. This argument is not supported with array inputs.
- **max_queue_size** – Used for generator or `keras.utils.Sequence` input only. Maximum size for the generator queue. If unspecified, `max_queue_size` will default to 10.
- **workers** – Used for generator or `keras.utils.Sequence` input only. Maximum number of processes to spin up when using process-based threading. If unspecified, `workers` will default to 1. If 0, will execute the generator on the main thread.
- **use_multiprocessing** – Used for generator or `keras.utils.Sequence` input only. If `True`, use process-based threading. If unspecified, `use_multiprocessing` will default to `False`. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

Returns `{test_metric: metric_value}` dictionary if the `metrics_names` of the model are specified, otherwise a list with the result test metrics (loss is usually the first value).

load(*model: str*, *reload: bool = False*) → Dict[str, Any]
(Re)-load a model from the file system.

Parameters

- **model** – Name of the model. It can be a folder name stored under `<workdir>/models`, or an absolute path to a model directory or file (Tensorflow directories, Protobuf models and HDF5 files are supported).
- **reload** – If `True`, the model will be reloaded from the filesystem even if it's been already loaded, otherwise the model currently in memory will be kept (default:

`False`).

Returns The model configuration.

predict(*model: str, inputs: Union[str, <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391754d50>, Iterable[T_co], Dict[str, Union[Iterable[T_co], <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391754d10>]]], batch_size: Optional[int] = None, verbose: int = 0, steps: Optional[int] = None, max_queue_size: int = 10, workers: int = 1, use_multiprocessing: bool = False*) → platypush.message.response.tensorflow.TensorflowPredictResponse

Generates output predictions for the input samples.

Parameters

- **model** – Name of the model. It can be a folder name stored under `<workdir>/models`, or an absolute path to a model directory or file (Tensorflow directories, Protobuf models and HDF5 files are supported).
- **inputs** – Input data. It can be:
  - A numpy array (or array-like), or a list of arrays in case the model has multiple inputs.
  - A TensorFlow tensor, or a list of tensors in case the model has multiple inputs.
  - A dict mapping input names to the corresponding array/tensors, if the model has named inputs.
  - A `tf.data` dataset. Should return a tuple of either `(inputs, targets)` or `(inputs, targets, sample_weights)`.
  - A generator or `keras.utils.Sequence` returning `(inputs, targets)` or `(inputs, targets, sample weights)`.
  - A string that points to a file. Supported formats:
    - CSV with header (`.csv` extension‘‘)
    - Numpy raw or compressed files (`.npy` or `.npz` extension)
    - Image files
    - An HTTP URL pointing to one of the file types listed above
- **batch_size** – Number of samples per gradient update. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of symbolic tensors, datasets, generators, or `keras.utils.Sequence` instances (since they generate batches).
- **verbose** – Verbosity mode, 0 or 1.
- **steps** – Total number of steps (batches of samples) before declaring the prediction round finished. Ignored with the default value of `None`. If x is a `tf.data` dataset and `steps` is None, `predict` will run until the input dataset is exhausted.
- **max_queue_size** – Integer. Used for generator or `keras.utils.Sequence` input only. Maximum size for the generator queue (default: 10).
- **workers** – Used for generator or `keras.utils.Sequence` input only. Maximum number of processes to spin up when using process-based threading. If unspecified, `workers` will default to 1. If 0, will execute the generator on the main thread.

- **use_multiprocessing** – Used for generator or `keras.utils.Sequence` input only. If `True`, use process-based threading. If unspecified, `use_multiprocessing` will default to `False`. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

Returns

platypush.message.response.tensorflow.TensorflowPredictResponse. Format:

- For regression models with no output labels specified: `outputs` will contain the output vector:

```
{
    "outputs": [[3.1415]]
}
```

- For regression models with output labels specified: `outputs` will be a list of `{label -> value}` maps:

```
{
    "outputs": [
        {
            "x": 42.0,
            "y": 43.0
        }
    ]
}
```

- For neural networks: `outputs` will contain the list of the output vector like in the case of regression, and `predictions` will store the list of `argmax` (i.e. the index of the output unit with the highest value) or their labels, if the model has output labels:

```
{
    "predictions": [
        "positive"
    ],
    "outputs": [
        {
            "positive": 0.998,
            "negative": 0.002
        }
    ]
}
```

remove (*model: str*) → None

Unload a module and, if stored on the filesystem, remove its resource files as well. WARNING: This operation is not reversible.

Parameters **model** – Name of the model.

save (*model: str*, *overwrite: bool = True*, ***opts*) → None

Save a model in memory to the filesystem. The model files will be stored under `<WORKDIR>/models/<model_name>`.

Parameters

- **model** – Model name.

- **overwrite** – Overwrite the model files if they already exist.
- **opts** – Extra options to be passed to `Model.save()`.

train(*model: str, inputs: Union[str, <sphinx.ext.autodoc.importer._MockObject object at 0x7fc39174d5d0>, Iterable[T_co], Dict[str, Union[Iterable[T_co], <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391ba6450>]]], outputs: Union[str, <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391ba6050>, Iterable[T_co], None] = None, batch_size: Optional[int] = None, epochs: int = 1, verbose: int = 1, validation_split: float = 0.0, validation_data: Optional[Tuple[Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fc391f85610>, Iterable[T_co]]]] = None, shuffle: Union[bool, str] = True, class_weight: Optional[Dict[int, float]] = None, sample_weight: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7fc3926337d0>, Iterable[T_co], None] = None, initial_epoch: int = 0, steps_per_epoch: Optional[int] = None, validation_steps: int = None, validation_freq: int = 1, max_queue_size: int = 10, workers: int = 1, use_multiprocessing: bool = False*) → platypush.message.response.tensorflow.TensorflowTrainResponse

Trains a model on a dataset for a fixed number of epochs.

Parameters

- **model** – Name of the model. It can be a folder name stored under `<workdir>/models`, or an absolute path to a model directory or file (Tensorflow directories, Protobuf models and HDF5 files are supported).
- **inputs** – Input data. It can be:
  - A numpy array (or array-like), or a list of arrays in case the model has multiple inputs.
  - A TensorFlow tensor, or a list of tensors in case the model has multiple inputs.
  - A dict mapping input names to the corresponding array/tensors, if the model has named inputs.
  - A `tf.data` dataset. Should return a tuple of either `(inputs, targets)` or `(inputs, targets, sample_weights)`.
  - A generator or `keras.utils.Sequence` returning `(inputs, targets)` or `(inputs, targets, sample weights)`.
  - A string that points to a file. Supported formats:
    - CSV with header (`.csv` extension``)
    - Numpy raw or compressed files (`.npy` or `.npz` extension)
    - Image files
    - An HTTP URL pointing to one of the file types listed above
    - Directories with images. If `inputs` points to a directory of images then the following conventions are followed:
      - The folder must contain exactly as many subfolders as the output units of your model. If the model has `output_labels` then those subfolders should be named as the output labels. Each subfolder will contain training examples that match the associated label (e.g. `positive` will contain all the positive images and `negative` all the negative images).
      - `outputs` doesn't have to be specified.

- **outputs** – Target data. Like the input data *x*, it can be a numpy array (or array-like) or TensorFlow tensor(s). It should be consistent with *x* (you cannot have Numpy inputs and tensor targets, or inversely). If *x* is a dataset, generator, or *keras.utils.Sequence* instance, *y* should not be specified (since targets will be obtained from *x*).
- **batch_size** – Number of samples per gradient update. If unspecified, `batch_size` will default to 32. Do not specify the `batch_size` if your data is in the form of symbolic tensors, datasets, generators, or `keras.utils.Sequence` instances (since they generate batches).
- **epochs** – Number of epochs to train the model. An epoch is an iteration over the entire `x` and `y` data provided. Note that in conjunction with `initial_epoch`, `epochs` is to be understood as "final epoch". The model is not trained for a number of iterations given by `epochs`, but merely until the epoch of index `epochs` is reached.
- **verbose** – Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch. Note that the progress bar is not particularly useful when logged to a file, so verbose=2 is recommended when not running interactively (eg, in a production environment).
- **validation_split** – Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the `x` and `y` data provided, before shuffling. Not supported when `x` is a dataset, generator or `keras.utils.Sequence` instance.
- **validation_data** – Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. `validation_data` will override `validation_split`. `validation_data` could be:
  - tuple `(x_val, y_val)` of arrays/numpy arrays/tensors
  - tuple `(x_val, y_val, val_sample_weights)` of Numpy arrays
  - dataset

  For the first two cases, `batch_size` must be provided. For the last case, `validation_steps` could be provided.
- **shuffle** – Boolean (whether to shuffle the training data before each epoch) or str (for 'batch'). 'batch' is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when `steps_per_epoch` is not `None`.
- **class_weight** – Optional dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function (during training only). This can be useful to tell the model to "pay more attention" to samples from an under-represented class.
- **sample_weight** – Optional iterable/numpy array of weights for the training samples, used for weighting the loss function (during training only). You can either pass a flat (1D) numpy array/iterable with the same length as the input samples (1:1 mapping between weights and samples), or in the case of temporal data, you can pass a 2D array with shape `(samples, sequence_length)`, to apply a different weight to every time step of every sample. In this case you should make sure to specify `sample_weight_mode="temporal"` in `compile()`. This

argument is not supported when `x` is a dataset, generator, or `keras.utils.Sequence` instance, instead provide the sample_weights as the third element of `x`.

- **`initial_epoch`** – Epoch at which to start training (useful for resuming a previous training run).
- **`steps_per_epoch`** – Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch. When training with input tensors such as TensorFlow data tensors, the default `None` is equal to the number of samples in your dataset divided by the batch size, or 1 if that cannot be determined. If x is a `tf.data` dataset, and 'steps_per_epoch' is None, the epoch will run until the input dataset is exhausted. This argument is not supported with array inputs.
- **`validation_steps`** – Only relevant if `validation_data` is provided and is a `tf.data` dataset. Total number of steps (batches of samples) to draw before stopping when performing validation at the end of every epoch. If 'validation_steps' is None, validation will run until the `validation_data` dataset is exhausted. In the case of a infinite dataset, it will run into a infinite loop. If 'validation_steps' is specified and only part of the dataset will be consumed, the evaluation will start from the beginning of the dataset at each epoch. This ensures that the same validation samples are used every time.
- **`validation_freq`** – Only relevant if validation data is provided. Integer or `collections_abc.Container` instance (e.g. list, tuple, etc.). If an integer, specifies how many training epochs to run before a new validation run is performed, e.g. `validation_freq=2` runs validation every 2 epochs. If a Container, specifies the epochs on which to run validation, e.g. `validation_freq=[1, 2, 10]` runs validation at the end of the 1st, 2nd, and 10th epochs.
- **`max_queue_size`** – Used for generator or `keras.utils.Sequence` input only. Maximum size for the generator queue. If unspecified, `max_queue_size` will default to 10.
- **`workers`** – Used for generator or `keras.utils.Sequence` input only. Maximum number of processes to spin up when using process-based threading. If unspecified, `workers` will default to 1. If 0, will execute the generator on the main thread.
- **`use_multiprocessing`** – Used for generator or `keras.utils.Sequence` input only. If `True`, use process-based threading. If unspecified, `use_multiprocessing` will default to `False`. Note that because this implementation relies on multiprocessing, you should not pass non-picklable arguments to the generator as they can't be passed easily to children processes.

Returns *`platypush.message.response.tensorflow.TensorflowTrainResponse`*

`unload`(*model: str*) → None

Remove a loaded model from memory.

Parameters **`model`** – Name of the model.

2.120 `platypush.plugins.todoist`

class `platypush.plugins.todoist.`**`TodoistPlugin`**(*api_token: str*, ***kwargs*)

Todoist integration.

Requires:

- **todoist-python** (`pip install todoist-python`)

You'll also need a Todoist token. You can get it *here <https://todoist.com/prefs/integrations>*.

`__init__`(*api_token: str*, ***kwargs*)

Parameters **`api_token`** – Todoist API token. You can get it *here <https://todoist.com/prefs/integrations>*.

`add_item`(*content: str*, *project_id: Optional[int] = None*, ***kwargs*)
Add a new item.

`archive`(*item_id: int*)
Archive an item by id.

`complete_item`(*item_id: int*)
Mark an item as done.

`delete_item`(*item_id: int*)
Delete an item by id.

`get_collaborators`() → platypush.message.response.todoist.TodoistCollaboratorsResponse
Get list of collaborators.

`get_filters`() → platypush.message.response.todoist.TodoistFiltersResponse
Get list of Todoist filters.

`get_items`() → platypush.message.response.todoist.TodoistItemsResponse
Get list of Todoist projects.

`get_live_notifications`() → platypush.message.response.todoist.TodoistLiveNotificationsResponse
Get list of Todoist live notifications.

`get_notes`() → platypush.message.response.todoist.TodoistNotesResponse
Get list of Todoist notes.

`get_project_notes`() → platypush.message.response.todoist.TodoistProjectNotesResponse
Get list of Todoist project notes.

`get_projects`() → platypush.message.response.todoist.TodoistProjectsResponse
Get list of Todoist projects.

`get_user`() → platypush.message.response.todoist.TodoistUserResponse
Get logged user info.

`sync`()
Sync/update info with the remote server.

`unarchive`(*item_id: int*)
Un-archive an item by id.

`uncomplete_item`(*item_id: int*)
Mark an item as not done.

`update_item`(*item_id: int*, ***kwargs*)
Update an item by id.

2.121 `platypush.plugins.torrent`

class `platypush.plugins.torrent.`**`TorrentPlugin`**(*download_dir=None*, *torrent_ports=None*, ***kwargs*)

Plugin to search and download torrents.

Requires:

- **python-libtorrent** (`pip install git+https://github.com/arvidn/libtorrent`)
- **requests** (`pip install requests`) [optional] for torrent info URL download

`__init__`(*download_dir=None*, *torrent_ports=None*, ***kwargs*)

Parameters

- **download_dir** (*str*) – Directory where the videos/torrents will be downloaded (default: none)
- **torrent_ports** (*list[int]*) – Torrent ports to listen on (default: 6881 and 6891)

`download`(*torrent*, *download_dir=None*, *_async=False*, *event_hndl=None*, *is_media=False*)

Download a torrent.

Parameters

- **torrent** (*str*) – Torrent to download. Supported formats:
  - Magnet URLs
  - Torrent URLs
  - Local torrent file
- **download_dir** (*str*) – Directory to download, overrides the default download_dir attribute (default: None)
- **_async** (*bool*) – If true then the method will add the torrent to the transfer and then return. Updates on the download status should be retrieved either by listening to torrent events or registering the event handler. If false (default) then the method will exit only when the torrent download is complete.
- **event_hndl** (*function*) – A function that takes an event object as argument and is invoked upon a new torrent event (download started, progressing, completed etc.)
- **is_media** (*bool*) – Set it to true if you're downloading a media file that you'd like to stream as soon as the first chunks are available. If so, then the events and the status method will only include media files

`pause`(*torrent*)

Pause/resume a torrent transfer.

Parameters **torrent** (*str*) – Torrent URL as returned from *get_status()*

`quit`()

Quits all the transfers and the active session

`remove`(*torrent*)

Stops and removes a torrent transfer.

Parameters **torrent** (*str*) – Torrent URL as returned from *get_status()*

`resume`(*torrent*)

Resume a torrent transfer.

Parameters **torrent** (*str*) – Torrent URL as returned from *get_status()*

search (*query*, *category=None*, **args*, ***kwargs*)
Perform a search of video torrents.

Parameters

- **query** (*str*) – Query string, video name or partial name
- **category** (*str or list*) – Category to search. Supported types: "movies", "tv", "anime". Default: None (search all categories)
- **language** (*str*) – Language code for the results - example: "en" (default: None, no filter)

status (*torrent=None*)
Get the status of the current transfers.

Parameters **torrent** (*str*) – Torrent path, URL or magnet URI whose status will be retrieve (default: None, retrieve all current transfers)

Returns A dictionary in the format torrent_url -> status

2.122 platypush.plugins.travisci

class platypush.plugins.travisci.**TravisciPlugin** (*token: str*, ***kwargs*)
Travis-Ci continuous integration plugin.

Setup:

- Get your API token from your Travis-Ci account settings page.

__init__ (*token: str*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

builds (*limit: int = 100*) → Dict[str, List[Dict[str, Any]]]
Get the list of builds triggered on the owned repositories

Parameters **limit** – Maximum number of builds to be retrieved (default: 100).

Returns Repo name -> List of builds

repos () → Dict[str, Dict[str, Any]]
Get the repos owned by current user. :return: Repo name -> Repo attributes mapping.

2.123 platypush.plugins.trello

class platypush.plugins.trello.**TrelloPlugin** (*api_key: str*, *api_secret: Optional[str] = None*, *token: Optional[str] = None*, ***kwargs*)
Trello integration.

Requires:

- **py-trello** (`pip install py-trello`)

You'll also need a Trello API key. You can get it *here <https://trello.com/app-key>*. You'll also need an auth token if you want to view/change private resources. You can generate a permanent token linked to your account on https://trello.com/1/connect?key=<KEY>&name=platypush&response_type=token&expiration=never&scope=read,write

__init__ (*api_key: str*, *api_secret: Optional[str] = None*, *token: Optional[str] = None*, ***kwargs*)

Parameters

- **`api_key`** – Trello API key. You can get it *here <https://trello.com/app-key>*.
- **`api_secret`** – Trello API secret. You can get it *here <https://trello.com/app-key>*.
- **`token`** – Trello token. It is required if you want to access or modify private resources. You can get a permanent token on https://trello.com/1/connect?key=<KEY>&name=platypush&response_type=token&expiration=never&scope=read,write

`add_card`(*board: str*, *list: str*, *name: str*, *description: Optional[str] = None*, *position: Optional[int] = None*, *labels: Optional[List[str]] = None*, *due: Union[datetime.datetime*, *str*, *None] = None*, *source: Optional[str] = None*, *assign: Optional[List[str]] = None*) → platypush.message.response.trello.TrelloCardResponse

Add a card to a list.

Parameters

- **`board`** – Board ID or name
- **`list`** – List ID or name
- **`name`** – Card name
- **`description`** – Card description
- **`position`** – Card position index
- **`labels`** – List of labels
- **`due`** – Due date (`datetime.datetime` object or ISO-format string)
- **`source`** – Card ID to clone from
- **`assign`** – List of assignee member IDs

`add_card_member`(*card_id: str*, *member_id: str*)

Add a member to a card.

Parameters

- **`card_id`** – Card ID
- **`member_id`** – Member ID

`add_checklist`(*card_id: str, title: str, items: List[str], states: Optional[List[bool]] = None*)

Add a checklist to a card.

Parameters

- **`card_id`** – Card ID
- **`title`** – Checklist title
- **`items`** – List of items in the checklist
- **`states`** – State of each item, True for checked, False for unchecked

`add_label`(*card_id: str*, *label: str*)

Add a label to a card.

Parameters

- **`card_id`** – Card ID
- **`label`** – Label name

`add_list` (*board: str*, *name: str*, *pos: Optional[int] = None*)
Add a list to a board.

Parameters

- **`board`** – Board ID or name
- **`name`** – List name
- **`pos`** – Optional position (default: last)

`add_member` (*board: str*, *member_id: str*, *member_type: str = 'normal'*)
Add a member to a board.

Parameters

- **`board`** – Board ID or name.
- **`member_id`** – Member ID to add.
- **`member_type`** – Member type - can be 'normal' or 'admin' (default: 'normal').

`archive_all_cards` (*board: str*, *list: str*)
Archive all the cards on a list.

Parameters

- **`board`** – Board ID or name
- **`list`** – List ID or name

`assign_card` (*card_id: str*, *member_id: str*)
Assign a card.

Parameters

- **`card_id`** – Card ID
- **`member_id`** – Member ID

`attach_card` (*card_id: str*, *name: Optional[str] = None*, *mime_type: Optional[str] = None*, *file: Optional[str] = None*, *url: Optional[str] = None*)
Add an attachment to a card. It can be either a local file or a remote URL.

Parameters

- **`card_id`** – Card ID
- **`name`** – File name
- **`mime_type`** – MIME type
- **`file`** – Path to the file
- **`url`** – URL to the file

`card_subscribe` (*card_id: str*)
Subscribe to a card. :param card_id: Card ID

`change_card_board` (*card_id: str*, *board: str*, *list: str = None*)
Move a card to a new board.

Parameters

- **`card_id`** – Card ID
- **`board`** – New board ID or name
- **`list`** – Optional target list ID or name

change_card_list (*card_id: str*, *list: str*)
Move a card to a new list.

Parameters

- **card_id** – Card ID
- **list** – List ID or name

change_card_pos (*card_id: str*, *position: int*)
Move a card to a new position.

Parameters

- **card_id** – Card ID
- **position** – New position index

close_board (*board: str*)
Close/archive a board.

Parameters **board** – Board ID or name

close_card (*card_id: str*)
Close/archive a card.

Parameters **card_id** – Card ID

close_list (*board: str*, *list: str*)
Close/archive a list.

Parameters

- **board** – Board ID or name
- **list** – List ID or name

comment_card (*card_id: str*, *text: str*)
Add a comment to a card.

Parameters

- **card_id** – Card ID
- **text** – Comment text

create_label (*board: str*, *name: str*, *color: Optional[str] = None*)
Add a label to a board.

Parameters

- **board** – Board ID or name
- **name** – Label name
- **color** – Optional HTML color

delete_card (*card_id: str*)
Permanently delete a card.

Parameters **card_id** – Card ID

delete_comment (*card_id: str*, *comment_id: str*)
Delete a comment.

Parameters

- **card_id** – Card ID

- **`comment_id`** – Comment ID

`delete_label`(*board: str*, *label: str*)
Delete a label from a board.

Parameters

- **`board`** – Board ID or name
- **`label`** – Label ID or name

`get_admin_members`(*board: str*) → platypush.message.response.trello.TrelloMembersResponse
Get the list of the admin members of a board. :param board: Board ID or name.

`get_board`(*board: str*) → platypush.message.response.trello.TrelloBoardResponse
Get the info about a board.

Parameters **`board`** – Board ID or name

`get_boards`(*all: bool = False*) → platypush.message.response.trello.TrelloBoardsResponse
Get the list of boards.

Parameters **`all`** – If True, return all the boards included those that have been closed/archived/deleted. Otherwise, only return open/active boards (default: False).

`get_cards`(*board: str*, *list: Optional[str] = None*, *all: bool = False*) → platypush.message.response.trello.TrelloCardsResponse
Get the list of cards on a board.

Parameters

- **`board`** – Board ID or name
- **`list`** – List ID or name. If set then the method will only return the cards found on that list (default: None)
- **`all`** – If True, return all the cards included those that have been closed/archived/deleted. Otherwise, only return open/active cards (default: False).

`get_lists`(*board: str*, *all: bool = False*) → platypush.message.response.trello.TrelloListsResponse
Get the list of lists on a board.

Parameters

- **`board`** – Board ID or name
- **`all`** – If True, return all the lists, included those that have been closed/archived/deleted. Otherwise, only return open/active lists (default: False).

`get_members`(*board: str*) → platypush.message.response.trello.TrelloMembersResponse
Get the list of all the members of a board. :param board: Board ID or name.

`list_subscribe`(*board: str*, *list: str*)
Subscribe to a list.

Parameters

- **`board`** – Board ID or name
- **`list`** – List ID or name

`list_unsubscribe`(*board: str*, *list: str*)
Unsubscribe from a list.

Parameters

- **`board`** – Board ID or name

- **`list`** – List ID or name

`move_all_cards`(*board: str*, *src: str*, *dest: str*)
Move all the cards from a list to another.

Parameters

- **`board`** – Board ID or name
- **`src`** – Source list
- **`dest`** – Target list

`move_list`(*board: str*, *list: str*, *position: int*)
Move a list to another position.

Parameters

- **`board`** – Board ID or name
- **`list`** – List ID or name
- **`position`** – New position index

`open_board`(*board: str*)
Re-open/un-archive a board.

Parameters **`board`** – Board ID or name

`open_card`(*card_id: str*)
Open/un-archive a card.

Parameters **`card_id`** – Card ID

`open_list`(*board: str*, *list: str*)
Open/un-archive a list.

Parameters

- **`board`** – Board ID or name
- **`list`** – List ID or name

`remove_attachment`(*card_id: str*, *attachment_id: str*)
Remove an attachment from a card.

Parameters

- **`card_id`** – Card ID
- **`attachment_id`** – Attachment ID

`remove_card_due`(*card_id: str*)
Remove the due date from a card.

Parameters **`card_id`** – Card ID

`remove_card_due_complete`(*card_id: str*)
Remove the due complete flag from a card.

Parameters **`card_id`** – Card ID

`remove_card_member`(*card_id: str*, *member_id: str*)
Remove a member from a card.

Parameters

- **`card_id`** – Card ID

- **`member_id`** – Member ID

`remove_label`(*card_id: str*, *label: str*)
Remove a label from a card.

Parameters

- **`card_id`** – Card ID
- **`label`** – Label name

`remove_member`(*board: str*, *member_id: str*)
Remove a member from a board.

Parameters

- **`board`** – Board ID or name.
- **`member_id`** – Member ID to remove.

`set_board_description`(*board: str*, *description: str*)
Change the description of a board.

Parameters

- **`board`** – Board ID or name.
- **`description`** – New description.

`set_board_name`(*board: str*, *name: str*)
Change the name of a board.

Parameters

- **`board`** – Board ID or name.
- **`name`** – New name.

`set_card_description`(*card_id: str*, *description: str*)
Change the description of a card.

Parameters

- **`card_id`** – Card ID
- **`description`** – New description

`set_card_due`(*card_id: str, due: Union[str, datetime.datetime]*)
Set the due date for a card.

Parameters

- **`card_id`** – Card ID
- **`due`** – Due date, as a datetime.datetime object or an ISO string

`set_card_due_complete`(*card_id: str*)
Set the due date of a card as completed.

Parameters **`card_id`** – Card ID

`set_card_name`(*card_id: str*, *name: str*)
Change the name of a card.

Parameters

- **`card_id`** – Card ID
- **`name`** – New name

`set_list_name`(*board: str*, *list: str*, *name: str*)
Change the name of a board list.

Parameters

- **`board`** – Board ID or name
- **`list`** – List ID or name
- **`name`** – New name

`unassign_card`(*card_id: str*, *member_id: str*)
Un-assign a card.

Parameters

- **`card_id`** – Card ID
- **`member_id`** – Member ID

`update_comment`(*card_id: str*, *comment_id: str*, *text: str*)
Update the content of a comment.

Parameters

- **`card_id`** – Card ID
- **`comment_id`** – Comment ID
- **`text`** – New comment text

2.124 `platypush.plugins.tts`

`class` `platypush.plugins.tts.`**`TtsPlugin`**(*language='en-gb'*, *media_plugin: Optional[str] = None*, *player_args: Optional[dict] = None*)
Default Text-to-Speech plugin. It leverages Google Translate.

Requires:

- At least a *media plugin* (see `platypush.plugins.media.MediaPlugin`) enabled/configured - used for speech playback.

`__init__`(*language='en-gb'*, *media_plugin: Optional[str] = None*, *player_args: Optional[dict] = None*)

Parameters

- **`language`** – Language code (default: `en-gb`).
- **`media_plugin`** – Media plugin to be used for audio playback. Supported:
  - `media.gstreamer`
  - `media.omxplayer`
  - `media.mplayer`
  - `media.mpv`
  - `media.vlc`
- **`player_args`** – Optional arguments that should be passed to the player plugin's `platypush.plugins.media.MediaPlugin.play()` method.

`say`(*text: str*, *language: Optional[str] = None*, *player_args: Optional[dict] = None*)
Say some text.

Parameters

- **`text`** – Text to say.
- **`language`** – Language code override.
- **`player_args`** – Optional arguments that should be passed to the player plugin's `platypush.plugins.media.MediaPlugin.play()` method.

2.125 `platypush.plugins.tts.google`

class `platypush.plugins.tts.google.`**`TtsGooglePlugin`**(*language: str = 'en-US', voice: Optional[str] = None, gender: str = 'FEMALE', credentials_file: str = '~/.credentials/platypush/google/platypush-tts.json', **kwargs*)

Advanced text-to-speech engine that leverages the Google Cloud TTS API. See https://cloud.google.com/text-to-speech/docs/quickstart-client-libraries#client-libraries-install-python for how to enable the API on your account and get your credentials.

Requires:

- **google-cloud-texttospeech** - `pip install google-cloud-texttospeech`
- **mplayer** - see your distribution docs on how to install the mplayer package

`__init__`(*language: str = 'en-US', voice: Optional[str] = None, gender: str = 'FEMALE', credentials_file: str = '~/.credentials/platypush/google/platypush-tts.json', **kwargs*)

Parameters

- **`language`** – Language code, see https://cloud.google.com/text-to-speech/docs/basics for supported languages
- **`voice`** – Voice type, see https://cloud.google.com/text-to-speech/docs/basics for supported voices
- **`gender`** – Voice gender (MALE, FEMALE or NEUTRAL)
- **`credentials_file`** – Where your GCloud credentials for TTS are stored, see https://cloud.google.com/text-to-speech/docs/basics
- **`kwargs`** – Extra arguments to be passed to the *`platypush.plugins.tts.TtsPlugin`* constructor.

`say`(*text: str, language: Optional[str] = None, voice: Optional[str] = None, gender: Optional[str] = None, player_args: Optional[dict] = None*)

Say a phrase.

Parameters

- **`text`** – Text to say.
- **`language`** – Language code override.
- **`voice`** – Voice type override.
- **`gender`** – Gender override.
- **`player_args`** – Optional arguments that should be passed to the player plugin's `platypush.plugins.media.MediaPlugin.play()` method.

2.126 `platypush.plugins.tv.samsung.ws`

class `platypush.plugins.tv.samsung.ws.`**`TvSamsungWsPlugin`**(*host: Optional[str] = None*, *port: int = 8002*, *timeout: Optional[int] = 5*, *name='platypush'*, *token_file: Optional[str] = None*, ***kwargs*)

Control a Samsung smart TV with Tizen OS over WiFi/ethernet. It should support any post-2016 Samsung with Tizen OS and enabled websocket-based connection.

Requires:

- **samsungtvws** (`pip install samsungtvws`)

`__init__`(*host: Optional[str] = None*, *port: int = 8002*, *timeout: Optional[int] = 5*, *name='platypush'*, *token_file: Optional[str] = None*, ***kwargs*)

Parameters

- **`host`** – IP address or host name of the smart TV.
- **`port`** – Websocket port (default: 8002).
- **`timeout`** – Connection timeout in seconds (default: 5, specify 0 or None for no timeout).
- **`name`** – Name of the remote device (default: platypush).
- **`token_file`** – Path to the token file (default: `~/.local/share/platypush/samsungtvws/token.txt`)

`back`(*host: Optional[str] = None*, *port: Optional[int] = None*) → None

Send back key to the device.

Parameters

- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`blue`(*host: Optional[str] = None*, *port: Optional[int] = None*) → None

Send blue key to the device.

Parameters

- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`channel`(*channel: int*, *host: Optional[str] = None*, *port: Optional[int] = None*) → None

Change to the selected channel.

Parameters

- **`channel`** – Channel index.
- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`channel_down`(*host: Optional[str] = None*, *port: Optional[int] = None*) → None

Send channel_down key to the device.

Parameters

- **`host`** – Default host IP/name override.

- **port** – Default port override.

channel_up (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send channel_up key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

close_app (*app_id: Union[int, str]*, *host: Optional[str] = None*, *port: Optional[int] = None*) → None
Close an app.

Parameters

- **app_id** – App ID.
- **host** – Default host IP/name override.
- **port** – Default port override.

device_info (*host: Optional[str] = None*, *port: Optional[int] = None*) → dict
Return the info of the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

digit (*digit: int*, *host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send a digit key to the device.

Parameters

- **digit** – Digit to send.
- **host** – Default host IP/name override.
- **port** – Default port override.

down (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send down key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

enter (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send enter key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

green (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send green key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

guide(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send guide key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

home(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send home key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

info(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send info key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

install_app(*app_id: Union[int, str], host: Optional[str] = None, port: Optional[int] = None*) → None
Install an app.

Parameters

- **app_id** – App ID.
- **host** – Default host IP/name override.
- **port** – Default port override.

left(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send left key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

list_apps(*host: Optional[str] = None*, *port: Optional[int] = None*) → list
Get the list of installed apps.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

menu(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send menu key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

mute(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send mute key to the device.

Parameters

- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`open_browser` (*url: str*, *host: Optional[str] = None*, *port: Optional[int] = None*) → None
Open a URL in the browser.

Parameters

- **`url`** – URL to open.
- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`power` (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send power on/off control to the device.

Parameters

- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`red` (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send red key to the device.

Parameters

- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`right` (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send right key to the device.

Parameters

- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`run_app` (*app_id: Union[int, str], host: Optional[str] = None, port: Optional[int] = None*) → None
Run an app by ID.

Parameters

- **`app_id`** – App ID.
- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`source` (*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send source key to the device.

Parameters

- **`host`** – Default host IP/name override.
- **`port`** – Default port override.

`status_app` (*app_id: Union[int, str], host: Optional[str] = None, port: Optional[int] = None*) → dict
Get the status of an app.

Parameters

- **app_id** – App ID.
- **host** – Default host IP/name override.
- **port** – Default port override.

tools(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send tools key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

up(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send up key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

volume_down(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send volume down control to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

volume_up(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send volume up control to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

yellow(*host: Optional[str] = None*, *port: Optional[int] = None*) → None
Send red key to the device.

Parameters

- **host** – Default host IP/name override.
- **port** – Default port override.

2.127 platypush.plugins.twilio

class platypush.plugins.twilio.**TwilioPhoneNumberType**
An enumeration.

class platypush.plugins.twilio.**TwilioPlugin**(*account_sid: str*, *auth_token: str*, *address_sid: Optional[str] = None*, *phone_number: Optional[str] = None*, *address_book: Optional[Dict[str, str]] = None*, ***kwargs*)
The Twilio plugin allows you to send messages and WhatsApp texts and make programmable phone call by using a Twilio account. Note that some features may require a Premium account.

Requires:

- **twilio** (`pip install twilio`)

`__init__`(*account_sid: str*, *auth_token: str*, *address_sid: Optional[str] = None*, *phone_number: Optional[str] = None*, *address_book: Optional[Dict[str, str]] = None*, ***kwargs*)

Parameters

- **`account_sid`** – Account SID.
- **`auth_token`** – Account authentication token.
- **`address_sid`** – SID of the default physical address - required to register a new number in some countries.
- **`phone_number`** – Default phone number associated to the account to be used for messages and calls.
- **`address_book`** – `name`->``phone_number`` mapping of contacts. You can use directly these names to send messages and make calls instead of the full phone number.

`create_address`(*customer_name: str*, *street: str*, *city: str*, *region: str*, *postal_code: str*, *iso_country: str*)

Create a new address associated to your account.

Parameters

- **`customer_name`** – Full name of the customer.
- **`street`** – Street name.
- **`city`** – City name.
- **`region`** – Region name.
- **`postal_code`** – Postal code.
- **`iso_country`** – ISO code of the country.

Returns

Details of the newly created address. Example:

```
{
  "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "city": "city",
  "customer_name": "customer_name",
  "date_created": "Tue, 18 Aug 2015 17:07:30 +0000",
  "date_updated": "Tue, 18 Aug 2015 17:07:30 +0000",
  "emergency_enabled": false,
  "friendly_name": null,
  "iso_country": "US",
  "postal_code": "postal_code",
  "region": "region",
  "sid": "ADXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "street": "street",
  "validated": false,
  "verified": false,
  "uri": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Addresses/
↪ADXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.json"
}
```

delete_message(*message_sid: str*)
Delete a send/received message.

Parameters **message_sid** – Message SID to be deleted.

get_available_phone_numbers(*country: str*, *number_type: str*) → List[dict]
Get a list of phone numbers of a certain type available for a certain country.

Parameters

- **country** – Country code (e.g. `US` or `NL`).
- **number_type** – Phone number type - e.g. `mobile`, `local` or `toll_free`.

Returns

A list of the available phone numbers with their properties and capabilities. Example:

```
[
  {
    "friendly_name": "+311234567890",
    "phone_number": "+311234567890",
    "lata": null,
    "rate_center": null,
    "latitude": null,
    "longitude": null,
    "locality": null,
    "region": null,
    "postal_code": null,
    "iso_country": "NL",
    "address_requirements": "any",
    "beta": false,
    "capabilities": {
      "voice": true,
      "SMS": true,
      "MMS": false,
      "fax": false
    }
  }
]
```

get_message(*message_sid: str*) → dict
Get the details of a stored message.

Parameters **message_sid** – Message SID to be retrieved.

Returns Message with its properties - see *send_message()*.

list_calls(*to: Optional[str] = None*, *from_: Optional[str] = None*, *parent_call_sid: Optional[str] = None*, *status: Optional[str] = None*, *start_time_before: Optional[str] = None*, *start_time: Optional[str] = None*, *start_time_after: Optional[str] = None*, *end_time_before: Optional[str] = None*, *end_time: Optional[str] = None*, *end_time_after: Optional[str] = None*, *limit: Optional[int] = None*, *page_size: Optional[int] = None*) → List[dict]
List the calls performed by the account, either the full list or those that match some filter.

Parameters

- **to** – Phone number or Client identifier of calls to include
- **from** – Phone number or Client identifier to filter *from* on
- **parent_call_sid** – Parent call SID to filter on

- **status** – The status of the resources to read
- **start_time_before** – Only include calls that started on this date
- **start_time** – Only include calls that started on this date
- **start_time_after** – Only include calls that started on this date
- **end_time_before** – Only include calls that ended on this date
- **end_time** – Only include calls that ended on this date
- **end_time_after** – Only include calls that ended on this date
- **limit** – Upper limit for the number of records to return. list() guarantees never to return more than limit. Default is no limit
- **page_size** – Number of records to fetch per request, when not set will use the default value of 50 records. If no page_size is defined but a limit is defined, list() will attempt to read the limit with the most efficient page size, i.e. min(limit, 1000)

Returns

A list of dictionaries, each representing the information of a call. Example:

```
[
    {
        "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "annotation": "billingreferencetag1",
        "answered_by": "machine_start",
        "api_version": "2010-04-01",
        "caller_name": "callerid1",
        "date_created": "Fri, 18 Oct 2019 17:00:00 +0000",
        "date_updated": "Fri, 18 Oct 2019 17:01:00 +0000",
        "direction": "outbound-api",
        "duration": "4",
        "end_time": "Fri, 18 Oct 2019 17:03:00 +0000",
        "forwarded_from": "calledvia1",
        "from": "+13051416799",
        "from_formatted": "(305) 141-6799",
        "group_sid": "GPXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "parent_call_sid": "CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "phone_number_sid": "PNXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "price": "-0.200",
        "price_unit": "USD",
        "sid": "CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "start_time": "Fri, 18 Oct 2019 17:02:00 +0000",
        "status": "completed",
        "subresource_uris": {
          "feedback": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Feedback.json",
          "feedback_summaries": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/FeedbackSummary.json
↪",
          "notifications": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Notifications.json",
          "recordings": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Recordings.json",
```

(continues on next page)

(continued from previous page)

```
        "payments": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Payments.json"
      },
      "to": "+13051913581",
      "to_formatted": "(305) 191-3581",
      "trunk_sid": "TKXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "uri": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.json",
      "queue_time": "1000"
  }
]
```

list_messages(*to: Optional[str] = None*, *from_: Optional[str] = None*, *date_sent_before: Optional[str] = None*, *date_sent: Optional[str] = None*, *date_sent_after: Optional[str] = None*, *limit: Optional[int] = None*, *page_size: Optional[int] = None*) → List[dict]

List all messages matching the specified criteria.

Parameters

- **to** – Recipient phone number or address book name.
- **from** – Sender phone number.
- **date_sent_before** – Maximum date filter (ISO format: YYYYMMDD with or without time).
- **date_sent** – Date filter (ISO format: YYYYMMDD with or without time).
- **date_sent_after** – Minimum date filter (ISO format: YYYYMMDD with or without time).
- **limit** – Maximum number of messages to be returned.
- **page_size** – Maximum number of messages per page.

Returns

List of selected messages. Example:

```
{
  "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "api_version": "2010-04-01",
  "body": "testing",
  "date_created": "Fri, 24 May 2019 17:44:46 +0000",
  "date_sent": "Fri, 24 May 2019 17:44:50 +0000",
  "date_updated": "Fri, 24 May 2019 17:44:50 +0000",
  "direction": "outbound-api",
  "error_code": null,
  "error_message": null,
  "from": "+12019235161",
  "messaging_service_sid": null,
  "num_media": "0",
  "num_segments": "1",
  "price": "-0.00750",
  "price_unit": "USD",
  "sid": "SMded05904ccb347238880ca9264e8fe1c",
  "status": "sent",
```

(continues on next page)

(continued from previous page)

```
  "subresource_uris": {
    "media": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Messages/
↪SMded05904ccb347238880ca9264e8fe1c/Media.json",
    "feedback": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Messages/
↪SMded05904ccb347238880ca9264e8fe1c/Feedback.json"
  },
  "to": "+18182008801",
  "uri": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Messages/
↪SMded05904ccb347238880ca9264e8fe1c.json"
}
```

make_call(*twiml: str, to: str, from_: Optional[str] = None, method: Optional[str] = None, status_callback: Optional[str] = None, status_callback_event: Optional[str] = None, status_callback_method: Optional[str] = None, fallback_url: Optional[str] = None, fallback_method: Optional[str] = None, send_digits: Optional[str] = None, timeout: Optional[int] = 30, record: bool = False, recording_channels: Optional[int] = None, recording_status_callback: Optional[str] = None, recording_status_callback_method: Optional[str] = None, recording_status_callback_event: Optional[str] = None, sip_auth_username: Optional[str] = None, sip_auth_password: Optional[str] = None, caller_id: Optional[str] = None, call_reason: Optional[str] = None*) → dict

Make an automated phone call from a registered Twilio number.

Parameters

- **twiml** – TwiML containing the logic to be executed in the call (see https://www.twilio.com/docs/voice/twiml).
- **to** – Recipient phone number or address book name.
- **from** – Registered Twilio phone number that will perform the call (default: default configured phone number).
- **method** – HTTP method to use to fetch TwiML if it's provided remotely.
- **status_callback** – The URL that should be called to send status information to your application.
- **status_callback_event** – The call progress events to be sent to the `status_callback` URL.
- **status_callback_method** – HTTP Method to use with status_callback.
- **fallback_url** – Fallback URL in case of error.
- **fallback_method** – HTTP Method to use with fallback_url.
- **send_digits** – The digits to dial after connecting to the number.
- **timeout** – Number of seconds to wait for an answer.
- **record** – Whether to record the call.
- **recording_channels** – The number of channels in the final recording.
- **recording_status_callback** – The URL that we call when the recording is available to be accessed.

- **recording_status_callback_method** – The HTTP method to use when calling the *recording_status_callback* URL.
- **recording_status_callback_event** – The recording status events that will trigger calls to the URL specified in *recording_status_callback*
- **sip_auth_username** – The username used to authenticate the caller making a SIP call.
- **sip_auth_password** – The password required to authenticate the user account specified in *sip_auth_username*.
- **caller_id** – The phone number, SIP address, or Client identifier that made this call. Phone numbers are in E.164 format (e.g., +16175551212). SIP addresses are formatted as *name@company.com*.
- **call_reason** – Reason for the call (Branded Calls Beta).

Returns

The call properties and details, as a dictionary. Example:

```
{
   "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
   "annotation": null,
   "answered_by": null,
   "api_version": "2010-04-01",
   "caller_name": null,
   "date_created": "Tue, 31 Aug 2010 20:36:28 +0000",
   "date_updated": "Tue, 31 Aug 2010 20:36:44 +0000",
   "direction": "inbound",
   "duration": "15",
   "end_time": "Tue, 31 Aug 2010 20:36:44 +0000",
   "forwarded_from": "+141586753093",
   "from": "+15017122661",
   "from_formatted": "(501) 712-2661",
   "group_sid": null,
   "parent_call_sid": null,
   "phone_number_sid": "PNXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
   "price": "-0.03000",
   "price_unit": "USD",
   "sid": "CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
   "start_time": "Tue, 31 Aug 2010 20:36:29 +0000",
   "status": "completed",
   "subresource_uris": {
     "notifications": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Notifications.json",
     "recordings": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Recordings.json",
     "feedback": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Feedback.json",
     "feedback_summaries": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/FeedbackSummary.json
↪",
     "payments": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Payments.json"
```

(continues on next page)

(continued from previous page)

```
    },
    "to": "+14155551212",
    "to_formatted": "(415) 555-1212",
    "trunk_sid": null,
    "uri": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Calls/
↪CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.json",
    "queue_time": "1000"
}
```

register_phone_number(*phone_number: str*, *friendly_name: Optional[str] = None*, *address_sid: Optional[str] = None*, *sms_url: Optional[str] = None*, *sms_fallback_url: Optional[str] = None*, *status_callback: Optional[str] = None*, *voice_caller_id_lookup: bool = True*, *voice_url: Optional[str] = None*, *voice_fallback_url: Optional[str] = None*, *area_code: Optional[str] = None*) → dict

Request to allocate a phone number on your Twilio account. The phone number should first be displayed as available in `get_available_phone_numbers()`.

Parameters

- **phone_number** – Phone number to be allocated.
- **friendly_name** – A string used to identify your new phone number.
- **address_sid** – Address SID. NOTE: some countries may require you to specify a valid address in order to register a new phone number (see meth:*create_address*). If none is specified then the configured `address_sid` (if available) will be applied.
- **sms_url** – URL to call when an SMS is received.
- **sms_fallback_url** – URL to call when an error occurs on SMS delivery/receipt.
- **status_callback** – URL to call when a status change occurs.
- **voice_caller_id_lookup** – Whether to perform ID lookup for incoming caller numbers.
- **voice_url** – URL to call when the number receives a call.
- **voice_fallback_url** – URL to call when a call fails.
- **area_code** – Override the area code for the new number.

Returns

Status of the newly created number. Example:

```
{
  "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "address_requirements": "none",
  "address_sid": "ADXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "api_version": "2010-04-01",
  "beta": false,
  "capabilities": {
    "voice": true,
    "sms": false,
    "mms": true,
```

(continues on next page)

(continued from previous page)

```
    "fax": false
  },
  "date_created": "Thu, 30 Jul 2015 23:19:04 +0000",
  "date_updated": "Thu, 30 Jul 2015 23:19:04 +0000",
  "emergency_status": "Active",
  "emergency_address_sid": "ADXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "friendly_name": "friendly_name",
  "identity_sid": "RIXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "origin": "origin",
  "phone_number": "+18089255327",
  "sid": "PNXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "sms_application_sid": "APXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "sms_fallback_method": "GET",
  "sms_fallback_url": "https://example.com",
  "sms_method": "GET",
  "sms_url": "https://example.com",
  "status_callback": "https://example.com",
  "status_callback_method": "GET",
  "trunk_sid": null,
  "uri": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/IncomingPhoneNumbers/
↪PNXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.json",
  "voice_application_sid": "APXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "voice_caller_id_lookup": false,
  "voice_fallback_method": "GET",
  "voice_fallback_url": "https://example.com",
  "voice_method": "GET",
  "voice_url": "https://example.com",
  "bundle_sid": "BUXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "voice_receive_mode": "voice",
  "status": "in-use"
}
```

send_message(*body: str*, *to: str*, *from_: Optional[str] = None*, *status_callback: Optional[str] = None*, *max_price: Optional[str] = None*, *attempt: Optional[int] = None*, *validity_period: Optional[int] = None*, *smart_encoded: bool = True*, *media_url: Optional[str] = None*) → dict

Send an SMS/MMS. Note: WhatsApp messages are also supported (and free of charge), although the functionality is currently quite limited. Full support is only available to WhatsApp Business profiles and indipendent software vendors approved by WhatsApp. If that's not the case, you can send WhatsApp messages through the Twilio Test account/number - as of now the `from_` field should be `whatsapp:+14155238886` and the `to` field should be `whatsapp:+<phone_number_here>`. More information here.

Parameters

- **body** – Message body.
- **to** – Recipient number or address book name.
- **from** – Sender number. If none is specified then the default configured `phone_number` will be used if available.
- **status_callback** – The URL to call to send status information to the application.
- **max_price** – The total maximum price up to 4 decimal places in US dollars acceptable for the message to be delivered.

- **attempt** – Total numer of attempts made , this inclusive to send out the message.
- **validity_period** – The number of seconds that the message can remain in our outgoing queue.
- **smart_encoded** – Whether to detect Unicode characters that have a similar GSM-7 character and replace them.
- **media_url** – The URL of the media to send with the message.

Returns

A mapping representing the status of the delivery. Example:

```
{
  "account_sid": "ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "api_version": "2010-04-01",
  "body": "Sent from your Twilio trial account - It works!",
  "date_created": "2020-08-17T16:32:09.341",
  "date_updated": "2020-08-17T16:32:09.526",
  "date_sent": null,
  "direction": "outbound-api",
  "error_code": null,
  "error_message": null,
  "from_": "+XXXXXXXXX",
  "messaging_service_sid": null,
  "num_media": "0",
  "num_segments": "1",
  "price": null,
  "price_unit": "USD",
  "sid": "XXXXXXXXXXXXX",
  "status": "queued",
  "subresource_uris": {
    "media": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Messages/
↪SMXXXXXXXXXXXXXXXXXXX/Media.json"
  },
  "to": "+XXXXXXXXXXXXXX",
  "uri": "/2010-04-01/Accounts/
↪ACXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX/Messages/
↪SMXXXXXXXXXXXXXXXXXXX.json"
}
```

update_message (*message_sid: str*, *body: str*) → dict

Update/redact the body of a message.

Parameters

- **message_sid** – Message SID to be updated.
- **body** – New message body.

Returns Updated message with its properties - see *send_message()*.

2.128 platypush.plugins.udp

class platypush.plugins.udp.**UdpPlugin**(***kwargs*)

Plugin for raw UDP communications.

send(*data: Union[bytes, str], host: str, port: int, binary: bool = False, timeout: Optional[float] = None, recv_response: bool = False, **recv_opts*)
Send data over a UDP connection.

Parameters

- **data** – Data to be sent, as bytes or string.
- **host** – Host IP/name.
- **port** – TCP port.
- **binary** – If set to True and `data` is a string then will be treated as base64-encoded binary input.
- **timeout** – Connection timeout in seconds (default: None).
- **recv_response** – If True then the action will wait for a response from the server before closing the connection. Note that `recv_opts` must be specified in this case - at least `length`.

2.129 `platypush.plugins.user`

class platypush.plugins.user.**UserPlugin**(***kwargs*)
Plugin to programmatically create and manage users and user sessions

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

authenticate_session(*session_token*)
Authenticate a session by token and return the associated user :return: dict.

Format:

```
{
    "user_id": int,
    "username": str,
    "created_at": str (in ISO format)
}
```

authenticate_user(*username, password*)
Authenticate a user :return: True if the provided username and password are correct, False otherwise

create_session(*username, password, expires_at=None*)
Create a user session :return: dict:

```
{
    "session_token": str,
    "user_id": int,
    "created_at": str (in ISO format),
    "expires_at": str (in ISO format),
}
```

create_user(*username, password, executing_user=None, executing_user_password=None, session_token=None, **kwargs*)

Create a user. This action needs to be executed by an already existing user, who needs to authenticate with
their own credentials, unless this is the first user created on the system.

Returns dict.

Format:

```
{
    "user_id": int,
    "username": str,
    "created_at": str (in ISO format)
}
```

delete_session(*session_token*)
Delete a user session

delete_user(*username*, *executing_user=None*, *executing_user_password=None*, *session_token=None*)
Delete a user

get_user_by_session(*session_token: str*) → dict
Get the user record associated to a session token.

Parameters **session_token** – Session token.

Returns

```
[
    {
        "user_id": 1,
        "username": "user1",
        "created_at": "2020-11-26T22:41:40.550574"
    }
]
```

get_users() → List[Dict[str, Any]]
Get the list of registered users. :return:

```
[
    {
        "user_id": 1,
        "username": "user1",
        "created_at": "2020-11-26T22:41:40.550574"
    },
    {
        "user_id": 2,
        "username": "user2",
        "created_at": "2020-11-28T21:10:23.224813"
    }
]
```

update_password(*username*, *old_password*, *new_password*)
Update the password of a user :return: True if the password was successfully updated, false otherwise

2.130 platypush.plugins.utils

class platypush.plugins.utils.**UtilsPlugin**(***kwargs*)
A plugin for general-purpose util methods

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

clear_interval(*name*)
Clear a running interval procedure

Parameters name (*str*) – Name of the interval to clear

clear_timeout(*name*)
Clear a pending timeout procedure

Parameters name (*str*) – Name of the timeout to clear

get_enabled_plugins() → dict

Returns The list of enabled plugins as a `name -> configuration` map.

get_interval(*name*)
Get info about a running interval

Parameters name (*str*) – Name of the interval to get

Returns dict. Example:

```
{
    "test_interval": {
        "seconds": 10.0,
        "actions": [
            {
                "action": "action_1",
                "args": {
                    "name_1": "value_1"
                }
            }
        ]
    }
}
```

If no such interval exist with the specified name then the value of the timeout name will be null.

get_intervals()
Get info about the running intervals

Returns dict

Example:

```
{
    "test_interval": {
        "seconds": 10.0,
        "actions": [
            {
                "action": "action_1",
                "args": {
                    "name_1": "value_1"
                }
            }
        ]
    }
}
```

get_sensor_plugins() → dict

Returns The list of enabled sensor plugins as a `name -> configuration` map.

get_switch_plugins() → dict

Returns The list of enabled switch plugins as a `name -> configuration` map.

get_timeout (*name*)

Get info about a pending timeout

Parameters **name** (*str*) – Name of the timeout to get

Returns dict

Example:

```
{
    "test_timeout": {
        "seconds": 10.0,
        "actions": [
            {
                "action": "action_1",
                "args": {
                    "name_1": "value_1"
                }
            }
        ]
    }
}
```

If no such timeout exist with the specified name then the value of the timeout name will be null.

get_timeouts ()

Get info about the pending timeouts

Returns dict.

Example:

```
{
    "test_timeout": {
        "seconds": 10.0,
        "actions": [
            {
                "action": "action_1",
                "args": {
                    "name_1": "value_1"
                }
            }
        ]
    }
}
```

set_interval (*seconds*, *actions*, *name=None*, ***args*)

Define a set of actions to run each specified amount of *seconds*.

Parameters

- **seconds** (*float*) – Number of seconds between two runs of the interval procedure
- **actions** (*list[dict]*) – List of actions to be executed at each interval
- **name** (*str*) – Set an optional name for this interval. It is advised to set a name if you are planning to programmatically cancel the interval in your business logic.
- **args** – Optional arguments/context to pass to the interval function

set_timeout (*seconds*, *actions*, *name=None*, ***args*)
Define a set of actions to run after the specified amount of *seconds*.

Parameters

- **seconds** (*float*) – Number of seconds before running the timeout procedure
- **actions** (*list[dict]*) – List of actions to be executed after the timeout expires
- **name** (*str*) – Set an optional name for this timeout. It is advised to set a name if you are planning to programmatically cancel the timeout in your business logic.
- **args** – Optional arguments/context to pass to the timeout function

sleep (*seconds*)
Make the current executor sleep for the specified number of seconds.

Parameters **seconds** (*float*) – Sleep seconds

2.131 platypush.plugins.variable

class `platypush.plugins.variable.`**VariablePlugin** (***kwargs*)
This plugin allows you to manipulate context variables that can be accessed across your tasks. It requires the *platypush.plugins.db* and *platypush.plugins.redis* plugins to be enabled, as the variables will be stored either persisted on a local database or on the local Redis instance.

Requires:

- **sqlalchemy** (`pip install sqlalchemy`)
- **redis** (`pip install redis`)

__init__ (***kwargs*)
The plugin will create a table named `variable` on the database configured in the *platypush.plugins.db* plugin. You'll have to specify a default `engine` in your `db` plugin configuration.

expire (*name*, *expire*)
Set a variable expiration on Redis

Parameters

- **name** (*str*) – Variable name
- **expire** (*int*) – Expiration time in seconds

get (*name*, *default_value=None*)
Get the value of a variable by name from the local db.

Parameters

- **name** (*str*) – Variable name
- **default_value** – What will be returned if the variable is not defined (default: None)

Returns A map in the format `{"<name>":"<value>"}`

mget (*name*)
Get the value of a variable by name from Redis.

Parameters **name** (*str*) – Variable name

Returns A map in the format `{"<name>":"<value>"}`

mset (***kwargs*)
Set a variable or a set of variables on Redis.

Parameters **kwargs** – Key-value list of variables to set (e.g. `foo='bar'`, `answer=42`)

Returns A map with the set variables

munset (*name*)
Unset a Redis variable by name if it's set

Parameters **name** (*str*) – Name of the variable to remove

set (***kwargs*)
Set a variable or a set of variables on the local db.

Parameters **kwargs** – Key-value list of variables to set (e.g. `foo='bar'`, `answer=42`)

unset (*name*)
Unset a variable by name if it's set on the local db.

Parameters **name** (*str*) – Name of the variable to remove

2.132 platypush.plugins.video.torrentcast

class platypush.plugins.video.torrentcast.**VideoTorrentcastPlugin** (*server='localhost'*, *port=9090*, **args*, ***kwargs*)

__init__ (*server='localhost'*, *port=9090*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

2.133 platypush.plugins.weather

class platypush.plugins.weather.**WeatherPlugin** (***kwargs*)
Base class for weather plugins.

2.134 platypush.plugins.weather.buienradar

class platypush.plugins.weather.buienradar.**WeatherBuienradarPlugin** (*lat: float*, *long: float*, *time_frame: int = 120*, ***kwargs*)

Plugin for getting weather updates through Buienradar - a Dutch weather app.

Requires:
- **buienradar** (`pip install buienradar`)

__init__ (*lat: float*, *long: float*, *time_frame: int = 120*, ***kwargs*)

Parameters

- **lat** – Default latitude
- **long** – Default longitude
- **time_frame** – Default number of minutes to look ahead for precipitation forecast

get_forecast(*lat: Optional[float] = None*, *long: Optional[float] = None*) → platypush.message.response.weather.buienradar.BuienradarForecastResponse

Get the weather forecast for the next days.

Parameters

- **lat** – Weather latitude (default: configured latitude)
- **long** – Weather longitude (default: configured longitude)

get_precipitation(*lat: Optional[float] = None*, *long: Optional[float] = None*, *time_frame: Optional[int] = None*) → platypush.message.response.weather.buienradar.BuienradarPrecipitationResponse

Get the precipitation forecast for the specified time frame.

Parameters

- **lat** – Weather latitude (default: configured latitude)
- **long** – Weather longitude (default: configured longitude)
- **time_frame** – Time frame for the forecast in minutes (default: configured time_frame)

get_weather(*lat: Optional[float] = None*, *long: Optional[float] = None*) → platypush.message.response.weather.buienradar.BuienradarWeatherResponse

Get the current weather conditions.

Parameters

- **lat** – Weather latitude (default: configured latitude)
- **long** – Weather longitude (default: configured longitude)

2.135 platypush.plugins.weather.darksky

class platypush.plugins.weather.darksky.**WeatherDarkskyPlugin**(*darksky_token*, *lat*, *long*, *units='si'*, ***kwargs*)

Plugin for getting weather updates through Darksky API.

NOTE: Shortly after being acquired by Apple, Darksky has [shut down their API](https://darksky.net/dev). If you have an API token already then it should keep working until the end of 2021, but no new signups are allowed - and yet again Apple hasn't lost a chance to stand against the developers.

Please use the *platypush.plugins.weather.openweathermap.WeatherOpenweathermapPlugin* plugin instead of this.

__init__(*darksky_token*, *lat*, *long*, *units='si'*, ***kwargs*)

Parameters

- **darksky_token** (*str*) – Your token for using the Darksky API, see https://darksky.net/dev
- **lat** (*float*) – Default forecast latitude

- **long** (*float*) – Default forecast longitude
- **units** – Weather units (default: "si").

Supported units:

- **si** (international system)
- **us** (US imperial units)
- **uk** (UK imperial units)
- **ca** (Canada imperial units)

get_current_weather (*lat=None*, *long=None*, ***kwargs*)

Get the current weather.

Parameters

- **lat** (*float*) – Weather latitude (default: configured latitude)
- **long** (*float*) – Weather longitude (default: configured longitude)

Returns A dictionary containing the current weather object.

Example output:

```
output = {
    "time": 1529947892,
    "summary": "Mostly Cloudy",
    "icon": "partly-cloudy-day",
    "precipIntensity": 0.0483,
    "precipProbability": 0.04,
    "precipType": "rain",
    "temperature": 27.94,
    "apparentTemperature": 29.6,
    "dewPoint": 20.01,
    "humidity": 0.62,
    "pressure": 1009.34,
    "windSpeed": 1.83,
    "windGust": 5.49,
    "windBearing": 192,
    "cloudCover": 0.66,
    "uvIndex": 0,
    "visibility": 16.09,
    "ozone": 273.74
}
```

get_daily_forecast (*lat=None*, *long=None*)

Get the daily forecast.

Parameters

- **lat** (*float*) – Weather latitude (default: configured latitude)
- **long** (*float*) – Weather longitude (default: configured longitude)

Returns A forecast object.

Example output:

```
"output": {
    "summary": "Light rain on Sunday, with high temperatures rising to 28°C␣
↪on Sunday.",
    "icon": "rain",
    "data": [
        {
            "time": 1529877600,
            "summary": "Mostly cloudy until afternoon.",
            "icon": "partly-cloudy-day",
            "sunriseTime": 1529896835,
            "sunsetTime": 1529957280,
            "moonPhase": 0.42,
            "precipIntensity": 0,
            "precipIntensityMax": 0.0051,
            "precipIntensityMaxTime": 1529888400,
            "precipProbability": 0,
            "temperatureHigh": 20.04,
            "temperatureHighTime": 1529931600,
            "temperatureLow": 10.68,
            "temperatureLowTime": 1529982000,
            "apparentTemperatureHigh": 20.04,
            "apparentTemperatureHighTime": 1529931600,
            "apparentTemperatureLow": 10.68,
            "apparentTemperatureLowTime": 1529982000,
            "dewPoint": 12.18,
            "humidity": 0.77,
            "pressure": 1025.16,
            "windSpeed": 3.84,
            "windGust": 6.51,
            "windGustTime": 1529881200,
            "windBearing": 336,
            "cloudCover": 0.5,
            "uvIndex": 6,
            "uvIndexTime": 1529928000,
            "visibility": 14.08,
            "ozone": 331.24,
            "temperatureMin": 13.89,
            "temperatureMinTime": 1529960400,
            "temperatureMax": 20.04,
            "temperatureMaxTime": 1529931600,
            "apparentTemperatureMin": 13.89,
            "apparentTemperatureMinTime": 1529960400,
            "apparentTemperatureMax": 20.04,
            "apparentTemperatureMaxTime": 1529931600
        },
        {
            "time": 1529964000,
            "summary": "Partly cloudy throughout the day.",
            "icon": "partly-cloudy-day",
            "sunriseTime": 1529983261,
            "sunsetTime": 1530043677,
            "moonPhase": 0.45,
            "precipIntensity": 0,
            "precipIntensityMax": 0,
            "precipProbability": 0,
            "temperatureHigh": 20.95,
            "temperatureHighTime": 1530018000,
```

(continues on next page)

(continued from previous page)

```
                "temperatureLow": 11.47,
                "temperatureLowTime": 1530064800,
                "apparentTemperatureHigh": 20.95,
                "apparentTemperatureHighTime": 1530018000,
                "apparentTemperatureLow": 11.47,
                "apparentTemperatureLowTime": 1530064800,
                "dewPoint": 10.19,
                "humidity": 0.69,
                "pressure": 1026.14,
                "windSpeed": 3.67,
                "windGust": 7.13,
                "windGustTime": 1530036000,
                "windBearing": 4,
                "cloudCover": 0.3,
                "uvIndex": 5,
                "uvIndexTime": 1530010800,
                "visibility": 16.09,
                "ozone": 328.59,
                "temperatureMin": 10.68,
                "temperatureMinTime": 1529982000,
                "temperatureMax": 20.95,
                "temperatureMaxTime": 1530018000,
                "apparentTemperatureMin": 10.68,
                "apparentTemperatureMinTime": 1529982000,
                "apparentTemperatureMax": 20.95,
                "apparentTemperatureMaxTime": 1530018000
            },
            # ...
        }
```

get_hourly_forecast (*lat=None*, *long=None*)

Get the hourly forecast.

Parameters

- **lat** (*float*) – Weather latitude (default: configured latitude)
- **long** (*float*) – Weather longitude (default: configured longitude)

Returns A forecast object.

Example output:

```
output = {
    "summary": "Partly cloudy starting tomorrow morning, continuing until␣
↪tomorrow evening.",
    "icon": "partly-cloudy-day",
    "data": [
        {
            "time": 1529946000,
            "summary": "Clear",
            "icon": "clear-day",
            "precipIntensity": 0,
            "precipProbability": 0,
            "temperature": 18.94,
            "apparentTemperature": 18.94,
            "dewPoint": 11.99,
            "humidity": 0.64,
```

(continues on next page)

(continued from previous page)

```
                "pressure": 1025.53,
                "windSpeed": 5.1,
                "windGust": 6.22,
                "windBearing": 329,
                "cloudCover": 0.14,
                "uvIndex": 1,
                "visibility": 14.19,
                "ozone": 334.3
            },
            {
                "time": 1529949600,
                "summary": "Clear",
                "icon": "clear-day",
                "precipIntensity": 0,
                "precipProbability": 0,
                "temperature": 18.41,
                "apparentTemperature": 18.41,
                "dewPoint": 11.12,
                "humidity": 0.63,
                "pressure": 1025.54,
                "windSpeed": 4.6,
                "windGust": 6.18,
                "windBearing": 340,
                "cloudCover": 0.07,
                "uvIndex": 1,
                "visibility": 16.09,
                "ozone": 333.53
            },
            # ...
}
```

2.136 `platypush.plugins.weather.openweathermap`

class `platypush.plugins.weather.openweathermap.`**`WeatherOpenweathermapPlugin`**(*token: str, location: Optional[str] = None, city_id: Optional[int] = None, lat: Optional[float] = None, long: Optional[float] = None, units: str = 'metric', **kwargs*)

`__init__`(*token: str, location: Optional[str] = None, city_id: Optional[int] = None, lat: Optional[float] = None, long: Optional[float] = None, units: str = 'metric', **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

`get`(*url, **kwargs*)

Perform a GET request

Parameters

- **`url`** (*`str`*) – Target URL
- **`kwargs`** (*`dict`*) – Additional arguments that will be transparently provided to the `requests` object, including but not limited to query params, data, JSON, headers etc. (see http://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

2.137 `platypush.plugins.websocket`

class `platypush.plugins.websocket.`**`WebsocketPlugin`**(**args, **kwargs*)

Plugin to send messages over a websocket connection

Requires:

- **websockets** (`pip install websockets`)

`__init__`(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

`send`(*url*, *msg*, *ssl_cert=None*, *ssl_key=None*, *ssl_cafile=None*, *ssl_capath=None*, **args*, ***kwargs*)
Sends a message to a websocket.

Parameters

- **`url`** – Websocket URL, e.g. ws://localhost:8765 or wss://localhost:8765
- **`msg`** – Message to be sent. It can be a list, a dict, or a Message object
- **`ssl_cert`** (*`str`*) – Path to the SSL certificate to be used, if the SSL connection requires client authentication as well (default: None)
- **`ssl_key`** (*`str`*) – Path to the SSL key to be used, if the SSL connection requires client authentication as well (default: None)
- **`ssl_cafile`** (*`str`*) – Path to the certificate authority file if required by the SSL configuration (default: None)
- **`ssl_capath`** (*`str`*) – Path to the certificate authority directory if required by the SSL configuration (default: None)

2.138 `platypush.plugins.wiimote`

class `platypush.plugins.wiimote.`**`WiimotePlugin`**(***kwargs*)
WiiMote plugin. A wrapper around the *`platypush.backend.wiimote`* backend to programmatically control a Nintendo WiiMote.

It requires the WiiMote backend to be enabled.

`close`()
Closes the connection with the WiiMote

`connect`()
Connects to the WiiMote

`rumble`(*secs*)
Rumbles the controller for the specified number of seconds

`set_leds`(*leds*)
Set the LEDs state on the controller

Parameters **`leds`** (*`list`*) – Iterable with the new states to be applied to the LEDs. Example: [1, 0, 0, 0] or (False, True, False, False)

`state`()
Return the state of the controller

2.139 `platypush.plugins.zeroconf`

class `platypush.plugins.zeroconf.`**`ZeroconfListener`**(*evt_queue: queue.Queue*)

`__init__`(*evt_queue: queue.Queue*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.plugins.zeroconf.**ZeroconfPlugin**(***kwargs*)

Plugin for Zeroconf services discovery.

Triggers:

- *platypush.message.event.zeroconf.ZeroconfServiceAddedEvent* when a new service is discovered.
- *platypush.message.event.zeroconf.ZeroconfServiceUpdatedEvent* when a service is updated.
- *platypush.message.event.zeroconf.ZeroconfServiceRemovedEvent* when a service is removed.

Requires:

- **zeroconf** (`pip install zeroconf`)

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

discover_service(*service: Union[str, list], timeout: Optional[int] = 5*) → Dict[str, Any]

Find all the services matching the specified type.

Parameters

- **service** – Service type (e.g. `_http._tcp.local.`) or list of service types.
- **timeout** – Browser timeout in seconds (default: 5). Specify None for no timeout - in such case the discovery will loop forever and generate events upon service changes.

Returns

A `service_type -> [service_names]` mapping. Example:

```
{
    "host1._platypush-http._tcp.local.": {
        "type": "_platypush-http._tcp.local.",
        "name": "host1._platypush-http._tcp.local.",
        "info": {
            "addresses": ["192.168.1.11"],
            "port": 8008,
            "host_ttl": 120,
            "other_ttl": 4500,
            "priority": 0,
            "properties": {
                "name": "Platypush",
                "vendor": "Platypush",
                "version": "0.13.2"
            },
            "server": "host1._platypush-http._tcp.local.",
            "weight": 0
        }
    }
}
```

get_services(*timeout: int = 5*) → List[str]

Get the full list of services found on the network.

Parameters **timeout** – Discovery timeout in seconds (default: 5).

Returns List of the services as strings.

2.140 `platypush.plugins.zigbee.mqtt`

class platypush.plugins.zigbee.mqtt.**ZigbeeMqttPlugin**(*host: str = 'localhost', port: int = 1883, base_topic: str = 'zigbee2mqtt', timeout: int = 10, tls_certfile: Optional[str] = None, tls_keyfile: Optional[str] = None, tls_version: Optional[str] = None, tls_ciphers: Optional[str] = None, username: Optional[str] = None, password: Optional[str] = None, **kwargs*)

This plugin allows you to interact with Zigbee devices over MQTT through any Zigbee sniffer and zigbee2mqtt.

In order to get started you'll need:

- A Zigbee USB adapter/sniffer (in this example I'll use the CC2531.
- A Zigbee debugger/emulator + downloader cable (only to flash the firmware).

Instructions:

- Install cc-tool either from sources or from a package manager.
- **Connect the Zigbee to your PC/RaspberryPi in this way:** USB -> CC debugger -> downloader cable -> CC2531 -> USB

  The debugger and the adapter should be connected *at the same time*. If the later `cc-tool` command throws up an error, put the device in sync while connected by pressing the _Reset_ button on the debugger.
- Check where the device is mapped. On Linux it will usually be `/dev/ttyACM0`.
- Download the latest Z-Stack firmware to your device. Instructions for a CC2531 device:

```
wget https://github.com/Koenkk/Z-Stack-firmware/raw/master/coordinator/Z-
↪Stack_Home_1.2/bin/default/CC2531_DEFAULT_20201127.zip
unzip CC2531_DEFAULT_20201127.zip
[sudo] cc-tool -e -w CC2531ZNP-Prod.hex
```

- You can disconnect your debugger and downloader cable once the firmware is flashed.
- Install `zigbee2mqtt`. First install a node/npm environment, then either install `zigbee2mqtt` manually or through your package manager. **NOTE**: many API breaking changes have occurred on Zigbee2MQTT 1.17.0, therefore this integration will only be compatible with the version 1.17.0 of the service or higher versions. Manual instructions:

```
# Clone zigbee2mqtt repository
[sudo] git clone https://github.com/Koenkk/zigbee2mqtt.git /opt/zigbee2mqtt
[sudo] chown -R pi:pi /opt/zigbee2mqtt  # Or whichever is your user

# Install dependencies (as user "pi")
cd /opt/zigbee2mqtt
npm install
```

- **You need to have an MQTT broker running somewhere. If not, you can install** Mosquitto through your package manager on any device in your network.
- Edit the `/opt/zigbee2mqtt/data/configuration.yaml` file to match the configuration of your MQTT broker:

```
# MQTT settings
mqtt:
    # MQTT base topic for zigbee2mqtt MQTT messages
    base_topic: zigbee2mqtt
    # MQTT server URL
```

(continues on next page)

(continued from previous page)

```
    server: 'mqtt://localhost'
    # MQTT server authentication, uncomment if required:
    # user: my_user
    # password: my_password
```

- Also make sure that `permit_join` is set to `True`, in order to allow Zigbee devices to join the network while you're configuring it. It's equally important to set `permit_join` to `False` once you have configured your network, to prevent accidental/malignant joins from outer Zigbee devices.
- Start the `zigbee2mqtt` daemon on your device (the official documentation also contains instructions on how to configure it as a `systemd` service:

```
cd /opt/zigbee2mqtt
npm start
```

- If you have Zigbee devices that are paired to other bridges, unlink them or do a factory reset to pair them to your new bridge.
- If it all goes fine, once the daemon is running and a new device is found you should see traces like this in the output of `zigbee2mqtt`:

```
zigbee2mqtt:info  2019-11-09T12:19:56: Successfully interviewed
↪'0x00158d0001dc126a', device has
successfully been paired
```

- You are now ready to use this integration.

Requires:

- **paho-mqtt** (`pip install paho-mqtt`)

`__init__`(*host: str = 'localhost'*, *port: int = 1883*, *base_topic: str = 'zigbee2mqtt'*, *timeout: int = 10*, *tls_certfile: Optional[str] = None*, *tls_keyfile: Optional[str] = None*, *tls_version: Optional[str] = None*, *tls_ciphers: Optional[str] = None*, *username: Optional[str] = None*, *password: Optional[str] = None*, ***kwargs*)

Parameters

- **`host`** – Default MQTT broker where `zigbee2mqtt` publishes its messages (default: `localhost`).
- **`port`** – Broker listen port (default: 1883).
- **`base_topic`** – Topic prefix, as specified in `/opt/zigbee2mqtt/data/configuration.yaml` (default: '`base_topic`').
- **`timeout`** – If the command expects from a response, then this timeout value will be used (default: 60 seconds).
- **`tls_cafile`** – If the connection requires TLS/SSL, specify the certificate authority file (default: None)
- **`tls_certfile`** – If the connection requires TLS/SSL, specify the certificate file (default: None)
- **`tls_keyfile`** – If the connection requires TLS/SSL, specify the key file (default: None)
- **`tls_version`** – If the connection requires TLS/SSL, specify the minimum TLS supported version (default: None)
- **`tls_ciphers`** – If the connection requires TLS/SSL, specify the supported ciphers (default: None)
- **`username`** – If the connection requires user authentication, specify the username (default: None)

- **password** – If the connection requires user authentication, specify the password (default: None)

bind_devices (*source: str*, *target: str*, ***kwargs*)
Bind two devices. Binding makes it possible that devices can directly control each other without the intervention of zigbee2mqtt or any home automation software. You may want to use this feature to bind for example an IKEA/Philips Hue dimmer switch to a light bulb, or a Zigbee remote to a thermostat. Read more on the zigbee2mqtt binding page.

Parameters

- **source** – Name of the source device. It can also be a group name, although the support is still experimental. You can also bind a specific device endpoint - for example `MySensor/temperature`.
- **target** – Name of the target device. You can also bind a specific device endpoint - for example `MyLight/state`.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

device_ban (*device: str*, ***kwargs*)
Ban a device from the network.

Parameters

- **device** – Display name of the device.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

device_check_ota_updates (*device: str*, ***kwargs*) → dict
Check if the specified device has any OTA updates available to install.

Parameters

- **device** – Address or friendly name of the device.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

Returns

```
{
    "id": "<device ID>",
    "update_available": true,
    "status": "ok"
}
```

device_get (*device: str*, *property: Optional[str] = None*, ***kwargs*) → Dict[str, Any]
Get the properties of a device. The returned keys vary depending on the device. For example, a light bulb may have the "`state`" and "`brightness`" properties, while an environment sensor may have the "`temperature`" and "`humidity`" properties, and so on.

Parameters

- **device** – Display name of the device.
- **property** – Name of the property that should be retrieved (default: all).
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

Returns Key->value map of the device properties.

device_install_ota_updates(*device: str*, ***kwargs*)
Install OTA updates for a device if available.

Parameters

- **device** – Address or friendly name of the device.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

device_remove(*device: str*, *force: bool = False*, ***kwargs*)
Remove a device from the network.

Parameters

- **device** – Display name of the device.
- **force** – Force the remove also if the removal wasn't acknowledged by the device. Note: a forced remove only removes the entry from the internal database, but the device is likely to connect again when restarted unless it's factory reset (default: False).
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

device_rename(*name: str*, *device: Optional[str] = None*, ***kwargs*)
Rename a device on the network.

Parameters

- **name** – New name.
- **device** – Current name of the device to rename. If no name is specified then the rename will affect the last device that joined the network.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

device_set(*device: str*, *property: str*, *value: Any*, ***kwargs*)
Set a properties on a device. The compatible properties vary depending on the device. For example, a light bulb may have the "`state`" and "`brightness`" properties, while an environment sensor may have the "`temperature`" and "`humidity`" properties, and so on.

Parameters

- **device** – Display name of the device.
- **property** – Name of the property that should be set.
- **value** – New value of the property.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

device_set_option(*device: str*, *option: str*, *value: Any*, ***kwargs*)
Change the options of a device. Options can only be changed, not added or deleted.

Parameters

- **device** – Display name of the device.
- **option** – Option name.
- **value** – New value.

- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

device_whitelist(*device: str*, ***kwargs*)
Whitelist a device on the network. Note: once at least a device is whitelisted, all the other non-whitelisted devices will be removed from the network.

Parameters

- **device** – Display name of the device.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

devices(***kwargs*) → List[Dict[str, Any]]
Get the list of devices registered to the service.

Parameters **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

Returns List of paired devices. Example output:

```
[
    {
        "date_code": "20190608",
        "friendly_name": "Coordinator",
        "ieee_address": "0x00123456789abcde",
        "network_address": 0,
        "supported": false,
        "type": "Coordinator",
        "interviewing": false,
        "interviewing_completed": true,
        "definition": null,
        "endpoints": {
            "13": {
                "bindings": [],
                "clusters": {
                    "input": ["genOta"],
                    "output": []
                },
                "output": []
            }
        }
    },

    {
        "date_code": "20180906",
        "friendly_name": "My Lightbulb",
        "ieee_address": "0x00123456789abcdf",
        "network_address": 52715,
        "power_source": "Mains (single phase)",
        "software_build_id": "5.127.1.26581",
        "model_id": "LCT001",
        "supported": true,
        "interviewing": false,
        "interviewing_completed": true,
        "type": "Router",
        "definition": {
            "description": "Hue white and color ambiance E26/E27/E14",
            "model": "9290012573A",
```

(continues on next page)

(continued from previous page)

```
            "vendor": "Philips",
            "exposes": [
                {
                    "features": [
                        {
                            "access": 7,
                            "description": "On/off state of this light",
                            "name": "state",
                            "property": "state",
                            "type": "binary",
                            "value_off": "OFF",
                            "value_on": "ON",
                            "value_toggle": "TOGGLE"
                        },
                        {
                            "access": 7,
                            "description": "Brightness of this light",
                            "name": "brightness",
                            "property": "brightness",
                            "type": "numeric",
                            "value_max": 254,
                            "value_min": 0
                        },
                        {
                            "access": 7,
                            "description": "Color temperature of this light",
                            "name": "color_temp",
                            "property": "color_temp",
                            "type": "numeric",
                            "unit": "mired",
                            "value_max": 500,
                            "value_min": 150
                        },
                        {
                            "description": "Color of this light in the CIE␣
↪1931 color space (x/y)",
                            "features": [
                                {
                                    "access": 7,
                                    "name": "x",
                                    "property": "x",
                                    "type": "numeric"
                                },
                                {
                                    "access": 7,
                                    "name": "y",
                                    "property": "y",
                                    "type": "numeric"
                                }
                            ],
                            "name": "color_xy",
                            "property": "color",
                            "type": "composite"
                        }
                    ],
                    "type": "light"
                },
```

(continues on next page)

(continued from previous page)

```
                {
                    "access": 2,
                    "description": "Triggers an effect on the light (e.g.␣
↪make light blink for a few seconds)",
                    "name": "effect",
                    "property": "effect",
                    "type": "enum",
                    "values": [
                        "blink",
                        "breathe",
                        "okay",
                        "channel_change",
                        "finish_effect",
                        "stop_effect"
                    ]
                },
                {
                    "access": 1,
                    "description": "Link quality (signal strength)",
                    "name": "linkquality",
                    "property": "linkquality",
                    "type": "numeric",
                    "unit": "lqi",
                    "value_max": 255,
                    "value_min": 0
                }
            ]
        },

        "endpoints": {
            "11": {
                "bindings": [],
                "clusters": {
                    "input": [
                        "genBasic",
                        "genIdentify",
                        "genGroups",
                        "genScenes",
                        "genOnOff",
                        "genLevelCtrl",
                        "touchlink",
                        "lightingColorCtrl",
                        "manuSpecificUbisysDimmerSetup"
                    ],
                    "output": [
                        "genOta"
                    ]
                },
                "configured_reportings": []
            },
            "242": {
                "bindings": [],
                "clusters": {
                    "input": [
                        "greenPower"
                    ],
                    "output": [
```

(continues on next page)

(continued from previous page)

```
                        "greenPower"
                    ]
                },
                "configured_reportings": []
            }
        }
    }
]
```

devices_get (*devices: Optional[List[str]] = None*, ***kwargs*) → Dict[str, dict]

Get the properties of the devices connected to the network.

Parameters

- **devices** – If set, then only the status of these devices (by friendly name) will be retrieved (default: retrieve all).
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

Returns

Key->value map of the device properties:

```
{
    "Bulb": {
        "state": "ON",
        "brightness": 254
    },
    "Sensor": {
        "temperature": 22.5
    }
}
```

factory_reset (***kwargs*)

Perform a factory reset of a device connected to the network, following the procedure required by the particular device (for instance, Hue bulbs require the Zigbee adapter to be close to the device while a button on the back of the bulb is pressed).

Parameters **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

group_add (*name: str*, *id: Optional[int] = None*, ***kwargs*)

Add a new group.

Parameters

- **name** – Display name of the group.
- **id** – Optional numeric ID (default: auto-generated).
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

group_add_device (*group: str*, *device: str*, ***kwargs*)

Add a device to a group.

Parameters

- **group** – Display name of the group.
- **device** – Display name of the device to be added.

- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

group_get (*group: str*, *property: Optional[str] = None*, ***kwargs*) → dict

Get one or more properties of a group. The compatible properties vary depending on the devices on the group. For example, a light bulb may have the "`state`" (with values `"ON"` and `"OFF"`) and "`brightness`" properties, while an environment sensor may have the "`temperature`" and "`humidity`" properties, and so on.

Parameters

- **group** – Display name of the group.
- **property** – Name of the property to retrieve (default: all available properties)
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

group_remove (*name: str*, ***kwargs*)

Remove a group.

Parameters

- **name** – Display name of the group.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

group_remove_device (*group: str*, *device: Optional[str] = None*, ***kwargs*)

Remove a device from a group.

Parameters

- **group** – Display name of the group.
- **device** – Display name of the device to be removed. If none is specified then all the devices registered to the specified group will be removed.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

group_rename (*name: str*, *group: str*, ***kwargs*)

Rename a group.

Parameters

- **name** – New name.
- **group** – Current name of the group to rename.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

group_set (*group: str*, *property: str*, *value: Any*, ***kwargs*)

Set a properties on a group. The compatible properties vary depending on the devices on the group. For example, a light bulb may have the "`state`" (with values `"ON"` and `"OFF"`) and "`brightness`" properties, while an environment sensor may have the "`temperature`" and "`humidity`" properties, and so on.

Parameters

- **group** – Display name of the group.
- **property** – Name of the property that should be set.
- **value** – New value of the property.

- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

groups(***kwargs*) → List[dict]
Get the groups registered on the device.

Parameters kwargs – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

info(***kwargs*) → dict
Get the information, configuration and state of the network.

Parameters kwargs – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

Returns

Example:

```
{
    "state": "online",
    "commit": "07cdc9d",
    "config": {
        "advanced": {
            "adapter_concurrent": null,
            "adapter_delay": null,
            "availability_blacklist": [],
            "availability_blocklist": [],
            "availability_passlist": [],
            "availability_timeout": 0,
            "availability_whitelist": [],
            "cache_state": true,
            "cache_state_persistent": true,
            "cache_state_send_on_startup": true,
            "channel": 11,
            "elapsed": false,
            "ext_pan_id": [
                221,
                221,
                221,
                221,
                221,
                221,
                221,
                221
            ],
            "homeassistant_discovery_topic": "homeassistant",
            "homeassistant_legacy_triggers": true,
            "homeassistant_status_topic": "hass/status",
            "last_seen": "disable",
            "legacy_api": true,
            "log_directory": "/opt/zigbee2mqtt/data/log/
↪%TIMESTAMP%",
            "log_file": "log.txt",
            "log_level": "debug",
            "log_output": [
                "console",
                "file"
            ],
            "log_rotation": true,
```

(continues on next page)

(continued from previous page)

```
            "log_syslog": {},
            "pan_id": 6754,
            "report": false,
            "soft_reset_timeout": 0,
            "timestamp_format": "YYYY-MM-DD HH:mm:ss"
        },
        "ban": [],
        "blocklist": [],
        "device_options": {},
        "devices": {
            "0x00123456789abcdf": {
                "friendly_name": "My Lightbulb"
            }
        },
        "experimental": {
            "output": "json"
        },
        "external_converters": [],
        "groups": {},
        "homeassistant": false,
        "map_options": {
            "graphviz": {
                "colors": {
                    "fill": {
                        "coordinator": "#e04e5d",
                        "enddevice": "#fff8ce",
                        "router": "#4ea3e0"
                    },
                    "font": {
                        "coordinator": "#ffffff",
                        "enddevice": "#000000",
                        "router": "#ffffff"
                    },
                    "line": {
                        "active": "#009900",
                        "inactive": "#994444"
                    }
                }
            }
        },
        "mqtt": {
            "base_topic": "zigbee2mqtt",
            "force_disable_retain": false,
            "include_device_information": false,
            "server": "mqtt://localhost"
        },
        "passlist": [],
        "permit_join": true,
        "serial": {
            "disable_led": false,
            "port": "/dev/ttyUSB0"
        },
        "whitelist": []
    },
    "coordinator": {
        "meta": {
            "maintrel": 3,
```

(continues on next page)

(continued from previous page)

```
                "majorrel": 2,
                "minorrel": 6,
                "product": 0,
                "revision": 20190608,
                "transportrev": 2
            },
            "type": "zStack12"
        },
        "log_level": "debug",
        "network": {
            "channel": 11,
            "extended_pan_id": "0xdddddddddddddddd",
            "pan_id": 6754
        },
        "permit_join": true,
        "version": "1.17.0"
}
```

log_level (*level: str*, ***kwargs*)
Change the log level at runtime. This change will not be persistent.

Parameters

- **level** – Possible values: 'debug', 'info', 'warn', 'error'.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

off (*device*, **args*, ***kwargs*) → dict
Implements `platypush.plugins.switch.plugin.SwitchPlugin.off()` and turns off a Zigbee device with a writable binary property.

on (*device*, **args*, ***kwargs*) → dict
Implements `platypush.plugins.switch.plugin.SwitchPlugin.on()` and turns on a Zigbee device with a writable binary property.

permit_join (*permit: bool = True*, *timeout: Optional[float] = None*, ***kwargs*)
Enable/disable devices from joining the network. This is not persistent (will not be saved to `configuration.yaml`).

Parameters

- **permit** – Set to True to allow joins, False otherwise.
- **timeout** – Allow/disallow joins only for this amount of time.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

status (*device: Optional[str] = None*, **args*, ***kwargs*)
Get the status of a device (by friendly name) or of all the connected devices (it wraps *`devices_get()`*).

Parameters **device** – Device friendly name (default: get all devices).

switches
Implements the *`platypush.plugins.switch.SwitchPlugin.switches`* property and returns the state of any device on the Zigbee network identified as a switch (a device is identified as a switch if it exposes a writable `state` property that can be set to `ON` or `OFF`).

toggle(*device*, **args*, ***kwargs*) → dict
Implements `platypush.plugins.switch.plugin.SwitchPlugin.toggle()` and toggles a Zigbee device with a writable binary property.

unbind_devices(*source: str*, *target: str*, ***kwargs*)
Un-bind two devices.

Parameters

- **source** – Name of the source device. You can also bind a specific device endpoint - for example `MySensor/temperature`.
- **target** – Name of the target device. You can also bind a specific device endpoint - for example `MyLight/state`.
- **kwargs** – Extra arguments to be passed to `platypush.plugins.mqtt.MqttPlugin.publish`()` (default: query the default configured device).

2.141 `platypush.plugins.zwave`

class `platypush.plugins.zwave.`**ZwavePlugin**(***kwargs*)
This plugin interacts with the devices on a Z-Wave network started through the *`platypush.backend.zwave.ZwaveBackend`* backend.

Requires:
- **python-openzwave** (`pip install python-openzwave`)
- The *`platypush.backend.zwave.ZwaveBackend`* backend configured and running.

activate_scene(*scene_id: Optional[int] = None*, *scene_label: Optional[str] = None*)
Activate a scene.

Parameters

- **scene_id** – Select by scene_id.
- **scene_label** – Select by scene label.

add_node(*do_security=False*)
Start the inclusion process to add a node to the network.

Parameters do_security – Whether to initialize the Network Key on the device if it supports the Security CC

add_node_to_group(*group_index: Optional[int] = None*, *group_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Add a node to a group.

Parameters

- **group_index** – Select group by group index.
- **group_label** – Select group by group label.
- **node_id** – Select node by node_id.
- **node_name** – Select node by node name.

Returns

cancel_command()
Cancel the current running command.

create_button(*button_id: Union[int, str], node_id: Optional[int] = None, node_name: Optional[str] = None*)
Create a handheld button on a device. Only intended for bridge firmware controllers.

Parameters

- **button_id** – The ID of the button.
- **node_id** – Filter by node_id.
- **node_name** – Filter by current node name.

create_new_primary()
Create a new primary controller on the network when the previous primary fails.

create_scene(*label: str*)
Create a new scene.

Parameters **label** – Scene label.

delete_button(*button_id: Union[int, str], node_id: Optional[int] = None, node_name: Optional[str] = None*)
Delete a button association from a device. Only intended for bridge firmware controllers.

Parameters

- **button_id** – The ID of the button.
- **node_id** – Filter by node_id.
- **node_name** – Filter by current node name.

get_battery_levels(*node_id: Optional[int] = None, node_name: Optional[str] = None*) → Dict[int, Any]
Get the battery levels of a node or of all the nodes on the network.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by name.

get_bulbs(*node_id: Optional[int] = None, node_name: Optional[str] = None*) → Dict[int, Any]
Get the bulbs/LEDs on the network or associated to a node.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by name.

get_capabilities() → List[str]
Get the capabilities of the controller.

get_dimmers(*node_id: Optional[int] = None, node_name: Optional[str] = None*) → Dict[int, Any]
Get the dimmers on the network or associated to a node.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by label.

get_doorlocks(*node_id: Optional[int] = None, node_name: Optional[str] = None*) → Dict[int, Any]
Get the doorlocks on the network or associated to a node.

Parameters

- **`node_id`** – Select node by node_id.
- **`node_name`** – Select node by name.

`get_groups`() → Dict[int, Any]
Get the groups on the network.

`get_node_config`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[int, Any]
Get the configuration values of a node or of all the nodes on the network.

Parameters

- **`node_id`** – Select node by node_id.
- **`node_name`** – Select node by label.

`get_node_stats`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[str, Any]
Get the statistics of a node on the network.

Parameters

- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by node name.

`get_nodes`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[str, Any]
Get the nodes associated to the network.

Parameters

- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by node name.

`get_power_levels`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[int, Any]
Get the power levels of this node.

Parameters

- **`node_id`** – Select node by node_id.
- **`node_name`** – Select node by name.

`get_protections`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[int, Any]
Get the protection-compatible devices on the network or associated to a node.

Parameters

- **`node_id`** – Select node by node_id.
- **`node_name`** – Select node by name.

`get_scene_values`(*scene_id: Optional[int] = None*, *scene_label: Optional[str] = None*) → dict
Get the values associated to a scene.

Parameters

- **`scene_id`** – Select by scene_id.
- **`scene_label`** – Select by scene label.

`get_scenes`() → Dict[str, Any]
Get the scenes configured on the network.

get_sensors(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[int, Any]
Get the sensors on the network or associated to a node.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by name.

get_switches(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[int, Any]
Get the switches on the network or associated to a node.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by name.

get_thermostats(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[int, Any]
Get the thermostats on the network or associated to a node.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by name.

get_usercodes(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[int, Any]
Get the usercodes on the network or associated to a node.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by name.

get_value(*value_id: Optional[int] = None*, *id_on_network: Optional[str] = None*, *value_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*) → Dict[str, Any]
Get a value on the network.

Parameters

- **value_id** – Select by value_id.
- **id_on_network** – Select value by id_on_network.
- **value_label** – Select value by [node_id/node_name, value_label]
- **node_id** – Select value by [node_id/node_name, value_label]
- **node_name** – Select value by [node_id/node_name, value_label]

hard_reset()
Perform a hard reset of the controller. It erases its network configuration settings. The controller becomes a primary controller ready to add devices to a new network.

heal(*refresh_routes: bool = False*)
Heal network by requesting nodes rediscover their neighbors.

Parameters **refresh_routes** – Whether to perform return routes initialization (default: `False`).

kill_command()
Immediately terminate any running command on the controller and release the lock.

node_add_value(*value_id: Optional[int] = None*, *id_on_network: Optional[str] = None*, *value_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Add a value to a node.

Parameters

- **value_id** – Select value by value_id.
- **id_on_network** – Select value by id_on_network.
- **value_label** – Select value by label.
- **node_id** – Select node by node_id.
- **node_name** – Select node by label.

node_heal(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*, *refresh_routes: bool = False*)
Heal network node by requesting the node to rediscover their neighbours.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by label.
- **refresh_routes** – Whether to perform return routes initialization. (default: `False`).

node_network_update(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Update the controller with network information.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by label.

node_refresh_info(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Fetch up-to-date information about the node.

Parameters

- **node_id** – Select node by node_id.
- **node_name** – Select node by label.

node_remove_value(*value_id: Optional[int] = None*, *id_on_network: Optional[str] = None*, *value_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Remove a value from a node.

Parameters

- **value_id** – Select value by value_id.
- **id_on_network** – Select value by id_on_network.
- **value_label** – Select value by [node_id/node_name, value_label]
- **node_id** – Select node by node_id.
- **node_name** – Select node by label.

node_update_neighbours(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Ask a node to update its neighbours table.

Parameters

- **`node_id`** – Select node by node_id.
- **`node_name`** – Select node by label.

`receive_configuration`()
Receive the configuration from the primary controller on the network. Requires a primary controller active.

`remove_failed_node`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Remove a failed node from the network.

Parameters

- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by node name.

`remove_node`()
Remove a node from the network.

`remove_node_from_group`(*group_index: Optional[int] = None*, *group_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Remove a node from a group.

Parameters

- **`group_index`** – Select group by group index.
- **`group_label`** – Select group by group label.
- **`node_id`** – Select node by node_id.
- **`node_name`** – Select node by node name.

Returns

`remove_scene`(*scene_id: Optional[int] = None*, *scene_label: Optional[str] = None*)
Remove a scene.

Parameters

- **`scene_id`** – Select by scene_id.
- **`scene_label`** – Select by scene label.

`replace_failed_node`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Replace a failed node on the network.

Parameters

- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by node name.

`replication_send`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Send node information from the primary to the secondary controller.

Parameters

- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by node name.

`request_network_update`(*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Request a network update to a node.

Parameters

- **node_id** – Filter by node_id.
- **node_name** – Filter by node name.

request_node_neighbour_update (*node_id: Optional[int] = None*, *node_name: Optional[str] = None*)

Request a neighbours list update to a node.

Parameters

- **node_id** – Filter by node_id.
- **node_name** – Filter by node name.

scene_add_value (*data: Optional[Any] = None*, *value_id: Optional[int] = None*, *id_on_network: Optional[str] = None*, *value_label: Optional[str] = None*, *scene_id: Optional[int] = None*, *scene_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)

Add a value to a scene.

WARNING: This method actually doesn't work, by own admission of the OpenZWave developer.

Parameters

- **data** – Data to set for the value (default: current value data).
- **value_id** – Select value by value_id.
- **id_on_network** – Select value by id_on_network.
- **value_label** – Select value by [node_id/node_name, value_label]
- **node_id** – Select value by [node_id/node_name, value_label]
- **node_name** – Select value by [node_id/node_name, value_label]
- **scene_id** – Select scene by scene_id.
- **scene_label** – Select scene by scene label.

scene_remove_value (*value_id: Optional[int] = None*, *id_on_network: Optional[str] = None*, *value_label: Optional[str] = None*, *scene_id: Optional[int] = None*, *scene_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)

Remove a value from a scene.

Parameters

- **value_id** – Select value by value_id.
- **id_on_network** – Select value by id_on_network.
- **value_label** – Select value by [node_id/node_name, value_label]
- **node_id** – Select value by [node_id/node_name, value_label]
- **node_name** – Select value by [node_id/node_name, value_label]
- **scene_id** – Select scene by scene_id.
- **scene_label** – Select scene by scene label.

set_controller_name (*name: str*)

Set the name of the controller on the network.

Parameters **name** – New controller name.

`set_node_location`(*location: str*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Set the location of a node.

Parameters

- **`location`** – Node location.
- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by current node name.

`set_node_manufacturer_name`(*manufacturer_name: str*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Set the manufacturer name of a node.

Parameters

- **`manufacturer_name`** – Manufacturer name.
- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by current node name.

`set_node_name`(*new_name: str*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Rename a node on the network.

Parameters

- **`new_name`** – New name for the node.
- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by current node name.

`set_node_product_name`(*product_name: str*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Set the product name of a node.

Parameters

- **`product_name`** – Product name.
- **`node_id`** – Filter by node_id.
- **`node_name`** – Filter by current node name.

`set_scene_label`(*new_label: str*, *scene_id: Optional[int] = None*, *scene_label: Optional[str] = None*)
Rename a scene/set the scene label.

Parameters

- **`new_label`** – New label.
- **`scene_id`** – Select by scene_id.
- **`scene_label`** – Select by current scene label.

`set_value`(*data*, *value_id: Optional[int] = None*, *id_on_network: Optional[str] = None*, *value_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)
Set a value.

Parameters

- **`data`** – Data to set for the value.

- **value_id** – Select value by value_id.
- **id_on_network** – Select value by id_on_network.
- **value_label** – Select value by [node_id/node_name, value_label]
- **node_id** – Select value by [node_id/node_name, value_label]
- **node_name** – Select value by [node_id/node_name, value_label]

set_value_label(*new_label: str*, *value_id: Optional[int] = None*, *id_on_network: Optional[str] = None*, *value_label: Optional[str] = None*, *node_id: Optional[int] = None*, *node_name: Optional[str] = None*)

Change the label/name of a value.

Parameters

- **new_label** – New value label.
- **value_id** – Select value by value_id.
- **id_on_network** – Select value by id_on_network.
- **value_label** – Select value by [node_id/node_name, value_label]
- **node_id** – Select value by [node_id/node_name, value_label]
- **node_name** – Select value by [node_id/node_name, value_label]

soft_reset()

Perform a soft reset of the controller. Resets a controller without erasing its network configuration settings.

status() → Dict[str, Any]

Get the status of the controller. :return: dict

switch_all(*state: bool*)

Switch all the connected devices on/off.

Parameters **state** – True (switch on) or False (switch off).

test(*count: int = 1*)

Send a number of test messages to every node and record results.

Parameters **count** – The number of test messages to send.

transfer_primary_role()

Add a new controller to the network and make it the primary. The existing primary will become a secondary controller.

write_config()

Store the current configuration of the network to the user directory.

CHAPTER 3

Events

3.1 platypush.message.event.adafruit

class platypush.message.event.adafruit.**ConnectedEvent**(**args*, ***kwargs*)
Event triggered when the backend connects to the Adafruit message queue

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.adafruit.**DisconnectedEvent**(**args*, ***kwargs*)
Event triggered when the backend disconnects from the Adafruit message queue

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.adafruit.**FeedUpdateEvent**(*feed*, *data*, **args*, ***kwargs*)
Event triggered upon Adafruit IO feed update

__init__(*feed*, *data*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.2 platypush.message.event.alarm

class platypush.message.event.alarm.**AlarmDismissedEvent**(*name: Optional[str] = None*, **args*, ***kwargs*)
Triggered when an alarm is dismissed.

class platypush.message.event.alarm.**AlarmEndedEvent**(*name: Optional[str] = None*, **args*, ***kwargs*)
Triggered when an alarm stops.

class platypush.message.event.alarm.**AlarmEvent**(*name: Optional[str] = None*, **args*, ***kwargs*)

__init__(*name: Optional[str] = None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.alarm.**AlarmSnoozedEvent**(*name: Optional[str] = None*, **args*, ***kwargs*)
Triggered when an alarm is snoozed.

class platypush.message.event.alarm.**AlarmStartedEvent**(*name: Optional[str] = None*, **args*, ***kwargs*)
Triggered when an alarm starts.

class platypush.message.event.alarm.**AlarmTimeoutEvent**(*name: Optional[str] = None*, **args*, ***kwargs*)
Triggered when an alarm times out.

3.3 platypush.message.event.application

class platypush.message.event.application.**ApplicationStartedEvent**(**args*, ***kwargs*)
Event triggered when the application has started and all the backends have been registered

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.4 platypush.message.event.assistant

class platypush.message.event.assistant.**AlarmEndEvent**(**args*, ***kwargs*)
Event triggered when an alarm ends on the assistant

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.assistant.**AlarmStartedEvent**(**args*, ***kwargs*)
Event triggered when an alarm starts on the assistant

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.assistant.**AlertEndEvent**(**args*, ***kwargs*)
Event triggered when an alert ends on the assistant

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.assistant.**AlertStartedEvent**(**args*, ***kwargs*)
Event triggered when an alert starts on the assistant

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.assistant.`**`AssistantEvent`**(*assistant=None*, **args*, ***kwargs*)
Base class for assistant events

`__init__`(*assistant=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.assistant.`**`ConversationEndEvent`**(*with_follow_on_turn=False*, **args*, ***kwargs*)
Event triggered when a conversation ends

`__init__`(*with_follow_on_turn=False*, **args*, ***kwargs*)

Parameters **`with_follow_on_turn`** (`str`) – Set to true if the conversation expects a user follow-up, false otherwise

class `platypush.message.event.assistant.`**`ConversationStartEvent`**(**args*, ***kwargs*)
Event triggered when a new conversation starts

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.assistant.`**`ConversationTimeoutEvent`**(**args*, ***kwargs*)
Event triggered when a conversation times out

`__init__`(**args*, ***kwargs*)

Parameters **`with_follow_on_turn`** (`str`) – Set to true if the conversation expects a user follow-up, false otherwise

class `platypush.message.event.assistant.`**`HotwordDetectedEvent`**(*hotword=None*, **args*, ***kwargs*)
Event triggered when a custom hotword is detected

`__init__`(*hotword=None*, **args*, ***kwargs*)

Parameters **`hotword`** (`str`) – The detected user hotword

class `platypush.message.event.assistant.`**`MicMutedEvent`**(*assistant=None*, **args*, ***kwargs*)
Event triggered when the microphone is muted.

class `platypush.message.event.assistant.`**`MicUnmutedEvent`**(*assistant=None*, **args*, ***kwargs*)
Event triggered when the microphone is muted.

class `platypush.message.event.assistant.`**`NoResponseEvent`**(**args*, ***kwargs*)
Event triggered when a conversation ends with no response

`__init__`(**args*, ***kwargs*)

Parameters **`with_follow_on_turn`** (`str`) – Set to true if the conversation expects a user follow-up, false otherwise

class `platypush.message.event.assistant.`**`ResponseEvent`**(*response_text*, **args*, ***kwargs*)
Event triggered when a response is processed by the assistant

__init__(*response_text*, **args*, ***kwargs*)

Parameters **response_text** (*str*) – Response text processed by the assistant

class platypush.message.event.assistant.**SpeechRecognizedEvent**(*phrase*, **args*, ***kwargs*)

Event triggered when a speech is recognized

__init__(*phrase*, **args*, ***kwargs*)

Parameters **phrase** (*str*) – Recognized user phrase

matches_condition(*condition*)

Overrides matches condition, and stops the conversation to prevent the default assistant response if the event matched some event hook condition

class platypush.message.event.assistant.**TimerEndEvent**(**args*, ***kwargs*)

Event triggered when a timer ends on the assistant

__init__(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.assistant.**TimerStartedEvent**(**args*, ***kwargs*)

Event triggered when a timer starts on the assistant

__init__(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.assistant.**VolumeChangedEvent**(*volume*, **args*, ***kwargs*)

Event triggered when the volume of the assistant changes

__init__(*volume*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.5 platypush.message.event.bluetooth

class platypush.message.event.bluetooth.**BluetoothConnectionRejectedEvent**(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Event triggered on bluetooth device connection rejected

__init__(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.bluetooth.**BluetoothDeviceConnectedEvent**(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Event triggered on bluetooth device connection

__init__(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.bluetooth.**BluetoothDeviceDisconnectedEvent**(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Event triggered on bluetooth device disconnection

__init__(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.bluetooth.**BluetoothDeviceFoundEvent**(*address: str*, *name: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a bluetooth device is found during a scan.

__init__(*address: str*, *name: Optional[str] = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.bluetooth.**BluetoothDeviceLostEvent**(*address: str*, *name: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a bluetooth device previously scanned is lost.

__init__(*address: str*, *name: Optional[str] = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.bluetooth.**BluetoothEvent**(*target=None*, *origin=None*, *id=None*, *timestamp=None*, *disable_logging=False*, *disable_web_clients_notification=False*, ***kwargs*)

class platypush.message.event.bluetooth.**BluetoothFileGetRequestEvent**(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Event triggered on bluetooth device file transfer get request

__init__(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.bluetooth.**BluetoothFilePutRequestEvent**(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Event triggered on bluetooth device file transfer put request

__init__(*address: str = None*, *port: str = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.bluetooth.**BluetoothFileReceivedEvent**(*path: str = None*, **args*, ***kwargs*)

Event triggered on bluetooth device file transfer put request

__init__(*path: str = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.6 platypush.message.event.button.flic

class platypush.message.event.button.flic.**FlicButtonEvent**(*btn_addr*, *sequence*, **args*, ***kwargs*)

Event triggered when a sequence of user short/long presses is detected on a Flic button (https://flic.io).

__init__(*btn_addr*, *sequence*, **args*, ***kwargs*)

Parameters

- **btn_addr** (*str*) – Physical address of the button that originated the event
- **sequence** (*list[str]*) – Detected sequence, as a list of Flic button event types (either "ShortPressEvent" or "LongPressEvent")

matches_condition(*condition*)

Parameters **condition** (*list*) – Condition to be checked against, as a sequence of button presses ("ShortPressEvent" and "LongPressEvent")

3.7 `platypush.message.event.camera`

class `platypush.message.event.camera.`**`CameraEvent`**(**args*, ***kwargs*)

Base class for camera events

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.camera.`**`CameraFrameCapturedEvent`**(*filename=None*, **args*, ***kwargs*)

Event triggered when a camera frame has been captured

`__init__`(*filename=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.camera.`**`CameraPictureTakenEvent`**(*filename=None*, **args*, ***kwargs*)

Event triggered when a snapshot has been taken

`__init__`(*filename=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.camera.`**`CameraRecordingStartedEvent`**(*device*, *filename=None*, **args*, ***kwargs*)

Event triggered when a new recording starts

`__init__`(*device*, *filename=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.camera.`**`CameraRecordingStoppedEvent`**(*device*, **args*, ***kwargs*)

Event triggered when a recording stops

`__init__`(*device*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.camera.`**`CameraVideoRenderedEvent`**(*filename=None*, **args*, ***kwargs*)

Event triggered when a sequence of frames has been rendered into a video

`__init__`(*filename=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.8 platypush.message.event.chat.telegram

class platypush.message.event.chat.telegram.**CommandMessageEvent**(*command: str*, *cmdargs: Optional[List[str]] = None*, **args*, ***kwargs*)

Event triggered when a new message is received by the Telegram bot.

__init__(*command: str*, *cmdargs: Optional[List[str]] = None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.chat.telegram.**ContactMessageEvent**(**args*, *message*, *user*, ***kwargs*)

class platypush.message.event.chat.telegram.**DocumentMessageEvent**(**args*, *message*, *user*, ***kwargs*)

class platypush.message.event.chat.telegram.**GroupChatCreatedEvent**(**args*, *message*, *user*, ***kwargs*)

class platypush.message.event.chat.telegram.**LocationMessageEvent**(**args*, *message*, *user*, ***kwargs*)

class platypush.message.event.chat.telegram.**MessageEvent**(**args*, *message*, *user*, ***kwargs*)

Event triggered when a new message is received by the Telegram bot.

__init__(**args*, *message*, *user*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.chat.telegram.**PhotoMessageEvent**(**args*, *message*, *user*, ***kwargs*)

class platypush.message.event.chat.telegram.**TelegramEvent**(**args*, *chat_id: int*, ***kwargs*)

__init__(**args*, *chat_id: int*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.chat.telegram.**TextMessageEvent**(**args*, *message*, *user*, ***kwargs*)

class platypush.message.event.chat.telegram.**VideoMessageEvent**(**args*, *message*, *user*, ***kwargs*)

3.9 platypush.message.event.clipboard

class platypush.message.event.clipboard.**ClipboardEvent**(*text: str*, **args*, ***kwargs*)

__init__(*text: str*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.10 `platypush.message.event.covid19`

class `platypush.message.event.covid19.`**`Covid19UpdateEvent`**(*confirmed: int*, *deaths: int*, *recovered: int*, *country: Optional[str] = None*, *country_code: Optional[str] = None*, *update_time: Optional[datetime.datetime] = None*, **args*, ***kwargs*)

__init__(*confirmed: int*, *deaths: int*, *recovered: int*, *country: Optional[str] = None*, *country_code: Optional[str] = None*, *update_time: Optional[datetime.datetime] = None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.11 `platypush.message.event.custom`

class `platypush.message.event.custom.`**`CustomEvent`**(*subtype: str*, **args*, ***kwargs*)
This type can be used to fire custom events upon which the user can implement custom hooks.

__init__(*subtype: str*, **args*, ***kwargs*)

Parameters

- **`subtype`** – This is the only mandatory attribute for this event type. It should be a string that unambiguously identifies a certain type of event (like `DISHWASHER_STARTED` or `SMOKE_DETECTED`).
- **`args`** – Extra list arguments for the event.
- **`kwargs`** – Extra key-value arguments for the event.

3.12 `platypush.message.event.distance`

class `platypush.message.event.distance.`**`DistanceSensorEvent`**(*distance: float*, *unit: str = 'mm'*, **args*, ***kwargs*)
Event triggered when a new value is processed by a distance sensor.

__init__(*distance: float*, *unit: str = 'mm'*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.13 `platypush.message.event.foursquare`

class `platypush.message.event.foursquare.`**`FoursquareCheckinEvent`**(*checkin: Dict[str, Any], *args, **kwargs*)

Event triggered when a new check-in occurs.

`__init__`(*checkin: Dict[str, Any], *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.14 `platypush.message.event.geo`

class `platypush.message.event.geo.`**`LatLongUpdateEvent`**(*latitude, longitude, altitude=None, *args, **kwargs*)

Event triggered upon GPS location update

`__init__`(*latitude, longitude, altitude=None, *args, **kwargs*)

Parameters

- **`latitude`** (*`float`*) – GPS latitude
- **`longitude`** (*`float`*) – GPS longitude
- **`altitude`** (*`float`*) – GPS altitude

3.15 `platypush.message.event.github`

class `platypush.message.event.github.`**`Actor`**(*id: str, login: str, display_login: str, url: str, gravatar_id: str, avatar_url: str*)

`__init__`(*id: str, login: str, display_login: str, url: str, gravatar_id: str, avatar_url: str*) → None

class `platypush.message.event.github.`**`GithubCommitCommentEvent`**(*payload: dict, *args, **kwargs*)

A commit comment is created.

`__init__`(*payload: dict, *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.github.`**`GithubCreateEvent`**(*payload: dict, *args, **kwargs*)

A git branch or tag is created.

`__init__`(*payload: dict, *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.github.`**`GithubDeleteEvent`**(*payload: dict, *args, **kwargs*)

A git branch or tag is deleted.

__init__(*payload: dict*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubEvent**(*event_type: str*, *created_at: datetime.datetime*, *actor: Optional[Dict[str, str]] = None*, *repo: Optional[Dict[str, str]] = None*, **args*, ***kwargs*)
Generic Github event

__init__(*event_type: str*, *created_at: datetime.datetime*, *actor: Optional[Dict[str, str]] = None*, *repo: Optional[Dict[str, str]] = None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubForkEvent**(*payload: dict*, **args*, ***kwargs*)
A user forks a watched repository.

__init__(*payload: dict*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubIssueCommentEvent**(*payload: dict*, **args*, ***kwargs*)
A comment is added or updated on an issue.

__init__(*payload: dict*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubIssueEvent**(*payload: dict*, **args*, ***kwargs*)
A new activity is registered on an issue.

__init__(*payload: dict*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubMemberEvent**(*payload: dict*, **args*, ***kwargs*)
New activity related to repository collaborators.

__init__(*payload: dict*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubPublicEvent**(*payload: dict*, **args*, ***kwargs*)
A private repository is made public.

__init__(*payload: dict*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubPullRequestEvent**(*payload: dict*, **args*, ***kwargs*)
New activity related to a pull request.

__init__(*payload: dict*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID

(default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubPullRequestReviewCommentEvent**(*payload: dict*, **args*, ***kwargs*)

New activity related to comments of a pull request.

__init__(*payload: dict*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubPushEvent**(*payload: dict*, **args*, ***kwargs*)

Github push event.

__init__(*payload: dict*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubReleaseEvent**(*payload: dict*, **args*, ***kwargs*)

New activity related to the release of a repository.

__init__(*payload: dict*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubSponsorshipEvent**(*payload: dict*, **args*, ***kwargs*)

New activity related to the sponsorship of a repository.

__init__(*payload: dict*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubWatchEvent**(*payload: dict*, **args*, ***kwargs*)

Event triggered when someone stars or starts watching a repository.

__init__(*payload: dict*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**GithubWikiEvent**(*payload: dict*, **args*, ***kwargs*)

A wiki page is created or updated on a watched repository.

__init__(*payload: dict*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.github.**Repo**(*id: str*, *name: str*, *url: str*)

__init__(*id: str*, *name: str*, *url: str*) → None

3.16 `platypush.message.event.google`

class `platypush.message.event.google.`**`GoogleDeviceEvent`**(*device_id*, *device_model_id=None*, **args*, ***kwargs*)

Base class for Google device events, see managing traits and handlers.

`__init__`(*device_id*, *device_model_id=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.google.`**`GoogleDeviceOnOffEvent`**(*on*, **args*, ***kwargs*)

Event triggered when a device receives an on/off command

`__init__`(*on*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.17 `platypush.message.event.google.fit`

class `platypush.message.event.google.fit.`**`GoogleFitEvent`**(*data_source_id*, *values*, **args*, ***kwargs*)

Event triggered upon new Google Fit data points

`__init__`(*data_source_id*, *values*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.18 `platypush.message.event.google.pubsub`

class `platypush.message.event.google.pubsub.`**`GooglePubsubMessageEvent`**(*topic: str*, *msg*, **args*, ***kwargs*)

Event triggered when a new message is received on a subscribed Google Pub/Sub topic.

`__init__`(*topic: str*, *msg*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.19 `platypush.message.event.gps`

class `platypush.message.event.gps.`**`GPSDeviceEvent`**(*path*, *activated=None*, *native=False*, *bps=None*, *parity=None*, *stopbits=None*, *cycle=None*, *driver=None*, **args*, ***kwargs*)

Event triggered when a new GPS device is connected or reconfigured

__init__(*path, activated=None, native=False, bps=None, parity=None, stopbits=None, cycle=None, driver=None, *args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.gps.`**GPSEvent**(**args, **kwargs*)
Generic class for GPS events

__init__(**args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.gps.`**GPSUpdateEvent**(*device=None, latitude=None, longitude=None, altitude=None, mode=None, epv=None, eph=None, sep=None, *args, **kwargs*)
Event triggered upon GPS status update

__init__(*device=None, latitude=None, longitude=None, altitude=None, mode=None, epv=None, eph=None, sep=None, *args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.gps.`**GPSVersionEvent**(*release=None, rev=None, proto_major=None, proto_minor=None, *args, **kwargs*)
Event usually triggered on startup or reconnection, when the GPS device advertises its version parameters

__init__(*release=None, rev=None, proto_major=None, proto_minor=None, *args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.20 platypush.message.event.http

class `platypush.message.event.http.`**HttpEvent**(*request, response, *args, **kwargs*)
Event triggered upon HTTP request/response cycle completion

__init__(*request, response, *args, **kwargs*)

Parameters

- **request** (`dict`) – Reference to the original HTTP request
- **response** (`dict or list`) – The server response

3.21 platypush.message.event.http.hook

class `platypush.message.event.http.hook.`**WebhookEvent**(**argv, hook, method, data=None, args=None, headers=None, **kwargs*)
Event triggered when a custom webhook is called.

__init__(**argv, hook, method, data=None, args=None, headers=None, **kwargs*)

Parameters

- **hook** (`str`) – Name of the invoked web hook, from http://host:port/hook/<hook>

- **method** (*str*) – HTTP method (in uppercase)
- **data** (*str or dict/list from JSON*) – Extra data passed over POST/PUT/DELETE
- **args** (*dict*) – Extra query string arguments
- **headers** – Request headers

3.22 platypush.message.event.http.ota.booking

class platypush.message.event.http.ota.booking.**NewReservationEvent**(*request*, *response*, **args*, ***kwargs*)

__init__(*request*, *response*, **args*, ***kwargs*)

Parameters

- **request** (*dict*) – Reference to the original HTTP request
- **response** (*dict or list*) – The server response

3.23 platypush.message.event.http.rss

class platypush.message.event.http.rss.**NewFeedEvent**(*request*, *response: list*, *source_id=None*, *source_title=None*, *source_url=None*, *title=None*, *digest_format=None*, *digest_filename=None*, **args*, ***kwargs*)

Event triggered when a monitored RSS feed has some new content

__init__(*request*, *response: list*, *source_id=None*, *source_title=None*, *source_url=None*, *title=None*, *digest_format=None*, *digest_filename=None*, **args*, ***kwargs*)

Parameters

- **request** – Original request
- **response** – Received response
- **source_id** – ID of the source that generated the event
- **source_title** – Name of the source, if available
- **source_url** – URL of the source
- **title** – Title of the new element
- **digest_format** – Format of the digest - either 'html' or 'pdf', if set
- **digest_filename** – File name of the digest, if it was dumped to file

3.24 `platypush.message.event.inotify`

class `platypush.message.event.inotify.`**`InotifyAccessEvent`**(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a monitored resource is accessed.

class `platypush.message.event.inotify.`**`InotifyCloseEvent`**(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a monitored resource is closed.

class `platypush.message.event.inotify.`**`InotifyCreateEvent`**(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a monitored resource is created.

class `platypush.message.event.inotify.`**`InotifyDeleteEvent`**(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a monitored resource is deleted.

class `platypush.message.event.inotify.`**`InotifyEvent`**(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Generic super-class for inotify events.

`__init__`(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **`path`** – Monitored path.
- **`resource`** – File/resource name.
- **`resource_type`** – INotify type of the resource, if available.

class `platypush.message.event.inotify.`**`InotifyModifyEvent`**(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a monitored resource is modified.

class `platypush.message.event.inotify.`**`InotifyMovedEvent`**(*path: str*, *old: Optional[str] = None*, *new: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a resource in a monitored path is moved.

`__init__`(*path: str*, *old: Optional[str] = None*, *new: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **`path`** – Monitored path.
- **`old`** – Old name.
- **`new`** – New name.

class `platypush.message.event.inotify.`**`InotifyOpenEvent`**(*path: str*, *resource: Optional[str] = None*, *resource_type: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a monitored resource is opened.

class `platypush.message.event.inotify.`**`InotifyPermissionsChangeEvent`**(*path: str*, *umask: int*, *resource: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when the permissions on a monitored resource are changed.

`__init__`(*path: str*, *umask: int*, *resource: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **`path`** – Monitored path.
- **`umask`** – New umask.
- **`resource`** – File/resource name.

3.25 `platypush.message.event.joystick`

class `platypush.message.event.joystick.`**`JoystickEvent`**(*code*, *state*, **args*, ***kwargs*)

Event triggered upon joystick event

`__init__`(*code*, *state*, **args*, ***kwargs*)

Parameters

- **`code`** (*`str`*) – Event code, usually the code of the source key/handle
- **`state`** (*`int`*) – State of the triggering element. Can be 0/1 for a button, -1/0/1 for an axis, a discrete integer for an analog input etc.

3.26 `platypush.message.event.kafka`

class `platypush.message.event.kafka.`**`KafkaMessageEvent`**(*msg*, **args*, ***kwargs*)

Kafka message event object. Fired when `platypush.backend.kafka` receives a new event.

`__init__`(*msg*, **args*, ***kwargs*)

Parameters **`msg`** (*`str or bytes stream`*) – Received message

3.27 platypush.message.event.light

class platypush.message.event.light.**LightAnimationStartedEvent**(**args*, *animation*, *lights: Optional[list] = None*, *groups: Optional[list] = None*, ***kwargs*)

Event triggered when a light animation is started.

__init__(**args*, *animation*, *lights: Optional[list] = None*, *groups: Optional[list] = None*, ***kwargs*)

Parameters **plugin_name** – Name of the `platypush.plugins.light.LightPlugin` instance that triggered the event.

class platypush.message.event.light.**LightAnimationStoppedEvent**(**args*, *animation=None*, *lights: Optional[list] = None*, *groups: Optional[list] = None*, ***kwargs*)

Event triggered when a light animation is stopped.

__init__(**args*, *animation=None*, *lights: Optional[list] = None*, *groups: Optional[list] = None*, ***kwargs*)

Parameters **plugin_name** – Name of the `platypush.plugins.light.LightPlugin` instance that triggered the event.

class platypush.message.event.light.**LightEvent**(**args*, *plugin_name: Optional[str] = None*, ***kwargs*)

Base class for light plugins events.

__init__(**args*, *plugin_name: Optional[str] = None*, ***kwargs*)

Parameters **plugin_name** – Name of the `platypush.plugins.light.LightPlugin` instance that triggered the event.

class platypush.message.event.light.**LightStatusChangeEvent**(*light_id=None*, *light_name=None*, *on=None*, *bri=None*, *sat=None*, *hue=None*, *ct=None*, *xy=None*, **args*, ***kwargs*)

Event triggered when the state of a lightbulb changes

__init__(*light_id=None*, *light_name=None*, *on=None*, *bri=None*, *sat=None*, *hue=None*, *ct=None*, *xy=None*, **args*, ***kwargs*)

Parameters

- **light_id** (*int*) – Light ID that triggered the event
- **light_name** (*str*) – Light name that triggered the event
- **on** (*bool*) – Set if the power state of the bulb changed
- **bri** (*int*) – Set if the brightness state of the bulb changed

- **sat** (*int*) – Set if the saturation state of the bulb changed
- **hue** (*int*) – Set if the hue state of the bulb changed
- **ct** (*int*) – Set if the color temperature state of the bulb changed
- **xy** (*list*) – Set if the color of the bulb (expressed in XY coordinates) has changed

3.28 `platypush.message.event.linode`

class `platypush.message.event.linode.`**LinodeEvent**(*target=None*, *origin=None*, *id=None*, *timestamp=None*, *disable_logging=False*, *disable_web_clients_notification=False*, ***kwargs*)

class `platypush.message.event.linode.`**LinodeInstanceStatusChanged**(*instance: str*, *status: str*, *old_status: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when the status of a Linode instance changes.

__init__(*instance: str*, *status: str*, *old_status: Optional[str] = None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.29 `platypush.message.event.mail`

class `platypush.message.event.mail.`**MailEvent**(*mailbox: str*, *message: Optional[platypush.plugins.mail.Mail] = None*, **args*, ***kwargs*)

__init__(*mailbox: str*, *message: Optional[platypush.plugins.mail.Mail] = None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.mail.`**MailFlaggedEvent**(*mailbox: str*, *message: Optional[platypush.plugins.mail.Mail] = None*, **args*, ***kwargs*)

Triggered when a message is marked as flagged/starred.

class `platypush.message.event.mail.`**MailReceivedEvent**(*mailbox: str*, *message: Optional[platypush.plugins.mail.Mail] = None*, **args*, ***kwargs*)

Triggered when a new email is received.

class `platypush.message.event.mail.`**MailSeenEvent**(*mailbox: str*, *message: Optional[platypush.plugins.mail.Mail] = None*, **args*, ***kwargs*)

Triggered when a previously unseen email is seen.

class `platypush.message.event.mail.`**`MailUnflaggedEvent`**(*mailbox: str*, *message: Optional[platypush.plugins.mail.Mail] = None*, **args*, ***kwargs*)

Triggered when a message previously marked as flagged/starred is unflagged.

3.30 `platypush.message.event.media`

class `platypush.message.event.media.`**`MediaEvent`**(*player=None*, *plugin=None*, **args*, ***kwargs*)

Base class for media events

`__init__`(*player=None*, *plugin=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.media.`**`MediaMuteChangedEvent`**(*mute*, *player=None*, *plugin=None*, **args*, ***kwargs*)

Event triggered when the media is muted/unmuted

`__init__`(*mute*, *player=None*, *plugin=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.media.`**`MediaPauseEvent`**(*player=None*, *plugin=None*, **args*, ***kwargs*)

Event triggered when a media playback is paused

`__init__`(*player=None*, *plugin=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.media.`**`MediaPlayEvent`**(*player=None*, *plugin=None*, *resource=None*, *title=None*, **args*, ***kwargs*)

Event triggered when a new media content is played

`__init__`(*player=None*, *plugin=None*, *resource=None*, *title=None*, **args*, ***kwargs*)

Parameters **`resource`** (`str`) – File name or URI of the played video

class `platypush.message.event.media.`**`MediaPlayRequestEvent`**(*player=None*, *plugin=None*, *resource=None*, *title=None*, **args*, ***kwargs*)

Event triggered when a new media playback request is received

`__init__`(*player=None*, *plugin=None*, *resource=None*, *title=None*, **args*, ***kwargs*)

Parameters **`resource`** (`str`) – File name or URI of the played video

class `platypush.message.event.media.`**`MediaSeekEvent`**(*position*, *player=None*, *plugin=None*, **args*, ***kwargs*)

Event triggered when the time position in the media changes

`__init__`(*position*, *player=None*, *plugin=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.media.**MediaStopEvent**(*player=None*, *plugin=None*, **args*, ***kwargs*)

Event triggered when a media is stopped

__init__(*player=None*, *plugin=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.media.**MediaVolumeChangedEvent**(*volume*, *player=None*, *plugin=None*, **args*, ***kwargs*)

Event triggered when the media volume changes

__init__(*volume*, *player=None*, *plugin=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.media.**NewPlayingMediaEvent**(*player=None*, *plugin=None*, *resource=None*, **args*, ***kwargs*)

Event triggered when a new media source is being played

__init__(*player=None*, *plugin=None*, *resource=None*, **args*, ***kwargs*)

Parameters **resource** (`str`) – File name or URI of the played resource

3.31 platypush.message.event.midi

class platypush.message.event.midi.**MidiMessageEvent**(*message*, *delay=None*, **args*, ***kwargs*)

Event triggered upon received MIDI message

__init__(*message*, *delay=None*, **args*, ***kwargs*)

Parameters

- **message** (`tuple[int]`) – Received MIDI message
- **delay** (`float`) – Time in seconds since the previous MIDI event (default: None)

3.32 platypush.message.event.mqtt

class platypush.message.event.mqtt.**MQTTMessageEvent**(*msg*, *host=None*, *port=None*, *topic=None*, **args*, ***kwargs*)

MQTT message event object. Fired when `platypush.backend.mqtt` receives a new event.

__init__(*msg*, *host=None*, *port=None*, *topic=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.33 `platypush.message.event.music`

class `platypush.message.event.music.`**`MusicEvent`**(*status*, *track*, *plugin_name=None*, **args*, ***kwargs*)

Base class for music events

`__init__`(*status*, *track*, *plugin_name=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.`**`MusicPauseEvent`**(**args*, ***kwargs*)

Event fired upon playback paused

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.`**`MusicPlayEvent`**(*status=None*, *track=None*, **args*, ***kwargs*)

Event fired upon music player playback start

`__init__`(*status=None*, *track=None*, **args*, ***kwargs*)

Parameters

- **`status`** (`dict`) – Player status
- **`track`** (`dict`) – Track being played

class `platypush.message.event.music.`**`MusicStopEvent`**(**args*, ***kwargs*)

Event fired upon playback stop

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.`**`MuteChangeEvent`**(*mute*, *status=None*, *track=None*, **args*, ***kwargs*)

Event fired upon mute change

`__init__`(*mute*, *status=None*, *track=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.`**`NewPlayingTrackEvent`**(*status=None*, *track=None*, **args*, ***kwargs*)

Event fired when a new track is being played

`__init__`(*status=None*, *track=None*, **args*, ***kwargs*)

Parameters

- **`status`** (`dict`) – Player status
- **`track`** (`dict`) – Track being played

class `platypush.message.event.music.`**`PlaybackConsumeModeChangeEvent`**(*state*, *status=None*, *track=None*, **args*, ***kwargs*)

Event fired upon consume mode change

__init__(*state*, *status=None*, *track=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.**PlaybackRandomModeChangeEvent**(*state*, *status=None*, *track=None*, **args*, ***kwargs*)
Event fired upon random mode change

__init__(*state*, *status=None*, *track=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.**PlaybackRepeatModeChangeEvent**(*state*, *status=None*, *track=None*, **args*, ***kwargs*)
Event fired upon repeat mode change

__init__(*state*, *status=None*, *track=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.**PlaybackSingleModeChangeEvent**(*state*, *status=None*, *track=None*, **args*, ***kwargs*)
Event fired upon single mode change

__init__(*state*, *status=None*, *track=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.**PlaylistChangeEvent**(*changes=None*, *status=None*, *track=None*, **args*, ***kwargs*)
Event fired upon playlist change

__init__(*changes=None*, *status=None*, *track=None*, **args*, ***kwargs*)

Parameters

- **changes** (*list*) – List with the tracks being added or removed
- **status** (*dict*) – Player status
- **track** (*dict*) – Track being played

class platypush.message.event.music.**SeekChangeEvent**(*position*, *status=None*, *track=None*, **args*, ***kwargs*)
Event fired upon seek change

__init__(*position*, *status=None*, *track=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.**VolumeChangeEvent**(*volume*, *status=None*, *track=None*, **args*, ***kwargs*)

Event fired upon volume change

__init__(*volume*, *status=None*, *track=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.34 platypush.message.event.music.snapcast

class platypush.message.event.music.snapcast.**ClientConnectedEvent**(*client*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon client connection

__init__(*client*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.snapcast.**ClientDisconnectedEvent**(*client*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon client disconnection

__init__(*client*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.snapcast.**ClientLatencyChangeEvent**(*client*, *latency*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon latency change on a client

__init__(*client*, *latency*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.music.snapcast.**ClientNameChangeEvent**(*client*, *name*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon name change of a client

__init__(*client*, *name*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.snapcast.`**`ClientVolumeChangeEvent`**(*client*, *volume*, *muted*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon volume change or mute status change on a client

`__init__`(*client*, *volume*, *muted*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.snapcast.`**`GroupMuteChangeEvent`**(*group*, *muted*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon mute status change

`__init__`(*group*, *muted*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.snapcast.`**`GroupStreamChangeEvent`**(*group*, *stream*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon group stream change

`__init__`(*group*, *stream*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.snapcast.`**`ServerUpdateEvent`**(*server*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon stream update

`__init__`(*server*, *host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.snapcast.`**`SnapcastEvent`**(*host='localhost'*, **args*, ***kwargs*)

Base class for Snapcast events

`__init__`(*host='localhost'*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.music.snapcast.`**`StreamUpdateEvent`**(*stream_id*, *stream*, *host='localhost'*, **args*, ***kwargs*)

Event fired upon stream update

__init__(*stream_id*, *stream*, *host='localhost'*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.35 `platypush.message.event.nextcloud`

class `platypush.message.event.nextcloud.`**NextCloudActivityEvent**(*activity_id: int*, *activity_type: str*, **args*, ***kwargs*)

__init__(*activity_id: int*, *activity_type: str*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.36 `platypush.message.event.nfc`

class `platypush.message.event.nfc.`**NFCDeviceConnectedEvent**(*reader=None*, **args*, ***kwargs*)
Event triggered when an NFC reader/writer devices is connected

__init__(*reader=None*, **args*, ***kwargs*)

Parameters **reader** (`str`) – Name or address of the reader that fired the event

class `platypush.message.event.nfc.`**NFCDeviceDisconnectedEvent**(*reader=None*, **args*, ***kwargs*)
Event triggered when an NFC reader/writer devices is disconnected

__init__(*reader=None*, **args*, ***kwargs*)

Parameters **reader** (`str`) – Name or address of the reader that fired the event

class `platypush.message.event.nfc.`**NFCEvent**(*reader=None*, *tag_id=None*, **args*, ***kwargs*)
Generic class for NFC events

__init__(*reader=None*, *tag_id=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.nfc.`**NFCTagDetectedEvent**(*reader=None*, *tag_id=None*, *records=None*, **args*, ***kwargs*)
Event triggered when an NFC tag is connected

__init__(*reader=None*, *tag_id=None*, *records=None*, **args*, ***kwargs*)

Parameters

- **reader** (`str`) – Name or address of the reader that fired the event
- **tag_id** (`str`) – ID of the NFC tag
- **records** (`str, bytes or JSON-serializable object`) – Optional, list of records read from the tag. If the tag contains JSON-serializable data then it will be cast by the backend into the appropriate object

class platypush.message.event.nfc.**NFCTagRemovedEvent**(*reader=None*, *tag_id=None*, **args*, ***kwargs*)
Event triggered when a NFC card is removed/disconnected

__init__(*reader=None*, *tag_id=None*, **args*, ***kwargs*)

Parameters

- **reader** (*str*) – Name or address of the reader that fired the event
- **tag_id** (*str*) – ID of the NFC tag

3.37 platypush.message.event.ping

class platypush.message.event.ping.**HostDownEvent**(*host: str*, **args*, ***kwargs*)
Event triggered when a remote host stops responding ping requests.

__init__(*host: str*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.ping.**HostUpEvent**(*host: str*, **args*, ***kwargs*)
Event triggered when a remote host starts responding ping requests.

__init__(*host: str*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.ping.**PingEvent**(*message=None*, **args*, ***kwargs*)
Ping event, used for testing purposes

__init__(*message=None*, **args*, ***kwargs*)

Parameters **message** (*object*) – Ping message

3.38 platypush.message.event.pushbullet

class platypush.message.event.pushbullet.**PushbulletEvent**(**args*, ***kwargs*)
PushBullet event object.

If you have configured the PushBullet backend with your account token, and enabled notification mirroring on the PushBullet app on your mobile devices, then the backend will trigger a PushbulletEvent whenever a new notiification hits your mobile, and you can react to that event through hooks that can, for example, log your notifications on a database, display them on a dashboard, let the built-in text-to-speech plugin read them out loud to you if they match the package name of your news app, display them on your smart watch if they are pictures, and so on.

__init__(**args*, ***kwargs*)
Platypush supports by default the PushBullet notification mirror format, https://docs.pushbullet.com/#mirrored-notifications

3.39 platypush.message.event.qrcode

class platypush.message.event.qrcode.**QrcodeEvent**(*target=None, origin=None, id=None, timestamp=None, disable_logging=False, disable_web_clients_notification=False, **kwargs*)

class platypush.message.event.qrcode.**QrcodeScannedEvent**(*results: List[platypush.message.response.qrcode.ResultMod *args, **kwargs*)

Event triggered when a QR-code or bar code is scanned.

__init__(*results: List[platypush.message.response.qrcode.ResultModel], *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.40 platypush.message.event.scard

class platypush.message.event.scard.**SmartCardDetectedEvent**(*atr, reader=None, *args, **kwargs*)

Event triggered when a smart card is detected

__init__(*atr, reader=None, *args, **kwargs*)

Parameters

- **atr** (*str*) – Smart card ATR (Answer To Reset)
- **reader** (*str*) – Name or address of the reader that fired the event

class platypush.message.event.scard.**SmartCardRemovedEvent**(*atr=None, reader=None, *args, **kwargs*)

Event triggered when a smart card is removed

__init__(*atr=None, reader=None, *args, **kwargs*)

Parameters

- **atr** (*str*) – Smart card ATR (Answer To Reset)
- **reader** (*str*) – Name or address of the reader that fired the event

3.41 platypush.message.event.sensor

class platypush.message.event.sensor.**SensorDataAboveThresholdEvent**(*data, *args, **kwargs*)

Event triggered when a sensor's read goes above a configured threshold

__init__(*data, *args, **kwargs*)

Parameters **data** – Sensor data

class `platypush.message.event.sensor.`**`SensorDataBelowThresholdEvent`**(*data*, **args*, ***kwargs*)

Event triggered when a sensor's read goes below a configured threshold

`__init__`(*data*, **args*, ***kwargs*)

Parameters **`data`** – Sensor data

class `platypush.message.event.sensor.`**`SensorDataChangeEvent`**(*data*, *source: Optional[str] = None*, **args*, ***kwargs*)

Event triggered when a sensor has new data

`__init__`(*data*, *source: Optional[str] = None*, **args*, ***kwargs*)

Parameters **`data`** – Sensor data

3.42 `platypush.message.event.sensor.ir`

class `platypush.message.event.sensor.ir.`**`IrKeyDownEvent`**(*message=None*, **args*, ***kwargs*)

Event triggered when a key on an infrared remote is pressed

`__init__`(*message=None*, **args*, ***kwargs*)

Parameters **`message`** – The received infrared message

class `platypush.message.event.sensor.ir.`**`IrKeyUpEvent`**(*message=None*, **args*, ***kwargs*)

Event triggered when a key on an infrared remote is released

`__init__`(*message=None*, **args*, ***kwargs*)

Parameters **`message`** – The received infrared message

class `platypush.message.event.sensor.ir.`**`IrSensorEvent`**(**args*, ***kwargs*)

Base class for infrared sensor events

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.43 `platypush.message.event.sensor.leap`

class `platypush.message.event.sensor.leap.`**`LeapConnectEvent`**(**args*, ***kwargs*)

Event triggered when a Leap Motion sensor is connected

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.sensor.leap.`**`LeapDisconnectEvent`**(**args*, ***kwargs*)

Event triggered when a Leap Motion sensor is disconnected

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.sensor.leap.**LeapFrameEvent**(*hands*, **args*, ***kwargs*)
Event triggered when a Leap Motion devices receives a new frame

__init__(*hands*, **args*, ***kwargs*)

Parameters **hands** (*dict*) – Reference to the detected hands properties (palm and fingers X,Y,Z position, direction etc.)

class platypush.message.event.sensor.leap.**LeapFrameStartEvent**(**args*, ***kwargs*)
Event triggered when a new sequence of frames is detected by the Leap Motion sensor

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.sensor.leap.**LeapFrameStopEvent**(**args*, ***kwargs*)
Event triggered when a Leap Sensor stops detecting frames

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.44 platypush.message.event.sensor.light

class platypush.message.event.sensor.light.**LightOffEvent**(**args*, ***kwargs*)
Event triggered when a light off event is detected

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.sensor.light.**LightOnEvent**(**args*, ***kwargs*)
Event triggered when a light on event is detected

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.45 platypush.message.event.serial

class platypush.message.event.serial.**SerialDataEvent**(*data*, *device=None*, **args*, ***kwargs*)
Event fired when a serial interface (generic USB, Arduino etc.) receives new data

__init__(*data*, *device=None*, **args*, ***kwargs*)

Parameters

- **data** (*object*) – Received data
- **device** (*str*) – Source device address or name

3.46 platypush.message.event.sound

class platypush.message.event.sound.**SoundEvent**(**args*, ***kwargs*)
Base class for sound events

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.sound.`**`SoundPlaybackPausedEvent`**(**args*, ***kwargs*)
Event triggered when the sound playback pauses

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.sound.`**`SoundPlaybackStartedEvent`**(*filename=None*, **args*, ***kwargs*)
Event triggered when a new sound playback starts

`__init__`(*filename=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.sound.`**`SoundPlaybackStoppedEvent`**(*filename=None*, **args*, ***kwargs*)
Event triggered when the sound playback stops

`__init__`(*filename=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.sound.`**`SoundRecordingPausedEvent`**(**args*, ***kwargs*)
Event triggered when a sound recording pauses

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.sound.`**`SoundRecordingStartedEvent`**(*filename=None*, **args*, ***kwargs*)
Event triggered when a new recording starts

`__init__`(*filename=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.sound.`**`SoundRecordingStoppedEvent`**(*filename=None*, **args*, ***kwargs*)
Event triggered when a sound recording stops

`__init__`(*filename=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.47 `platypush.message.event.stt`

class `platypush.message.event.stt.`**`ConversationDetectedEvent`**(*speech: str*, **args*, ***kwargs*)
Event triggered when speech is detected after a hotword.

class `platypush.message.event.stt.`**`HotwordDetectedEvent`**(*hotword: str = ''*, **args*, ***kwargs*)

Event triggered when a custom hotword is detected.

`__init__`(*hotword: str = ''*, **args*, ***kwargs*)

Parameters **`hotword`** – The detected user hotword.

class `platypush.message.event.stt.`**`SpeechDetectedEvent`**(*speech: str*, **args*, ***kwargs*)

Event triggered when speech is detected.

`__init__`(*speech: str*, **args*, ***kwargs*)

Parameters **`speech`** – Speech detected, as a string

class `platypush.message.event.stt.`**`SpeechDetectionStartedEvent`**(**args*, ***kwargs*)

Event triggered when the speech detection engine starts.

class `platypush.message.event.stt.`**`SpeechDetectionStoppedEvent`**(**args*, ***kwargs*)

Event triggered when the speech detection engine stops.

class `platypush.message.event.stt.`**`SpeechStartedEvent`**(**args*, ***kwargs*)

Event triggered when speech starts being detected.

class `platypush.message.event.stt.`**`SttEvent`**(**args*, ***kwargs*)

Base class for speech-to-text events

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.48 `platypush.message.event.tensorflow`

class `platypush.message.event.tensorflow.`**`TensorflowBatchEndedEvent`**(*batch: int*, **args*, ***kwargs*)

Triggered when a the processing of a Tensorflow model training/evaluation batch ends.

`__init__`(*batch: int*, **args*, ***kwargs*)

Parameters **`batch`** – Batch index.

class `platypush.message.event.tensorflow.`**`TensorflowBatchStartedEvent`**(*batch: int*, **args*, ***kwargs*)

Triggered when a Tensorflow model training/evaluation batch starts being processed.

`__init__`(*batch: int*, **args*, ***kwargs*)

Parameters **`batch`** – Batch index.

class `platypush.message.event.tensorflow.`**`TensorflowEpochEndedEvent`**(*epoch: int*, **args*, ***kwargs*)

Triggered when a Tensorflow model training/evaluation epoch ends.

`__init__`(*epoch: int*, **args*, ***kwargs*)

Parameters **`epoch`** – Epoch index.

class `platypush.message.event.tensorflow.`**`TensorflowEpochStartedEvent`**(*epoch: int*, **args*, ***kwargs*)

Triggered when a Tensorflow model training/evaluation epoch begins.

`__init__`(*epoch: int*, **args*, ***kwargs*)

Parameters **`epoch`** – Epoch index.

class `platypush.message.event.tensorflow.`**`TensorflowEvent`**(*model: str, logs: Optional[Dict[str, Union[int, float]]], *args, **kwargs*)

`__init__`(*model: str, logs: Optional[Dict[str, Union[int, float]]], *args, **kwargs*)

Parameters

- **`model`** – Name of the Tensorflow model.
- **`logs`** – Logs and metrics.

class `platypush.message.event.tensorflow.`**`TensorflowTrainEndedEvent`**(*model: str, logs: Optional[Dict[str, Union[int, float]]], *args, **kwargs*)

Triggered when the training phase of a Tensorflow model ends.

class `platypush.message.event.tensorflow.`**`TensorflowTrainStartedEvent`**(*model: str, logs: Optional[Dict[str, Union[int, float]]], *args, **kwargs*)

Triggered when a Tensorflow model starts being trained.

3.49 `platypush.message.event.todoist`

class `platypush.message.event.todoist.`**`CheckedItemEvent`**(*item*, **args*, ***kwargs*)

Event triggered when an item is checked.

`__init__`(*item*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.todoist.`**`ItemContentChangeEvent`**(*item*, **args*, ***kwargs*)

Event triggered when the content of an item changes.

`__init__`(*item*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.todoist.**ModifiedItemEvent**(*item*, **args*, ***kwargs*)
Event triggered when an item is changed.

__init__(*item*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.todoist.**NewItemEvent**(*item*, **args*, ***kwargs*)
Event triggered when a new item is created.

__init__(*item*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.todoist.**RemovedItemEvent**(*item*, **args*, ***kwargs*)
Event triggered when a new item is removed.

__init__(*item*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.todoist.**TodoistEvent**(*target=None*, *origin=None*, *id=None*, *timestamp=None*, *disable_logging=False*, *disable_web_clients_notification=False*, ***kwargs*)

class platypush.message.event.todoist.**TodoistSyncRequiredEvent**(*target=None*, *origin=None*, *id=None*, *timestamp=None*, *disable_logging=False*, *disable_web_clients_notification=False*, ***kwargs*)
Event triggered when an event occurs that doesn't fall into the categories above.

3.50 platypush.message.event.torrent

class platypush.message.event.torrent.**TorrentDownloadCompletedEvent**(*url*, **args*, ***kwargs*)
Event triggered upon torrent state change

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentDownloadProgressEvent**(*url*, **args*, ***kwargs*)
Event triggered upon torrent download progress

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentDownloadStartEvent**(*url*, **args*, ***kwargs*)
Event triggered upon torrent download start

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentDownloadStopEvent**(*url*, **args*, ***kwargs*)
Event triggered when a torrent transfer is stopped

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentDownloadedMetadataEvent**(*url*, **args*, ***kwargs*)
Event triggered upon torrent metadata download completed

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentEvent**(**args*, ***kwargs*)
Base class for torrent events

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentPausedEvent**(*url*, **args*, ***kwargs*)
Event triggered when a torrent transfer is paused

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentQueuedEvent**(*url*, **args*, ***kwargs*)
Event triggered upon when a new torrent transfer is queued

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentRemovedEvent**(*url*, **args*, ***kwargs*)
Event triggered when a torrent transfer is removed.

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentResumedEvent**(*url*, **args*, ***kwargs*)
Event triggered when a torrent transfer is resumed

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.torrent.**TorrentSeedingStartEvent**(*url*, **args*, ***kwargs*)
Event triggered upon torrent seeding start

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.torrent.`**`TorrentStateChangeEvent`**(*url*, **args*, ***kwargs*)

Event triggered upon torrent state change

__init__(*url*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.51 `platypush.message.event.travisci`

class `platypush.message.event.travisci.`**`TravisciBuildEvent`**(*repository_id: int, repository_name: str, repository_slug: str, passed: bool, build_id: int, build_number: int, duration: int, previous_state: str, private: bool, tag: Optional[str], branch: str, commit_id: Optional[str], commit_sha: Optional[str], commit_message: Optional[str], committed_at: str, created_by: str, started_at: str, finished_at: str, *args, **kwargs*)

__init__(*repository_id: int, repository_name: str, repository_slug: str, passed: bool, build_id: int, build_number: int, duration: int, previous_state: str, private: bool, tag: Optional[str], branch: str, commit_id: Optional[str], commit_sha: Optional[str], commit_message: Optional[str], committed_at: str, created_by: str, started_at: str, finished_at: str, *args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.travisci.`**`TravisciBuildFailedEvent`**(**args*, ***kwargs*)

Event triggered when a Travis-Ci build fails.

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.travisci.`**`TravisciBuildPassedEvent`**(**args*, ***kwargs*)

Event triggered when a Travis-Ci build passes.

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.52 `platypush.message.event.trello`

class `platypush.message.event.trello.`**`ArchivedCardEvent`**(**args*, ***kwargs*)
Event triggered when a card is archived.

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.trello.`**`CardEvent`**(*card_id: str*, *card_name: str*, *list_id: str*, *list_name: str*, *board_id: str*, *board_name: str*, *closed: bool*, *member_id: str*, *member_username: str*, *member_fullname: str*, *date: datetime.datetime*, **args*, ***kwargs*)

`__init__`(*card_id: str*, *card_name: str*, *list_id: str*, *list_name: str*, *board_id: str*, *board_name: str*, *closed: bool*, *member_id: str*, *member_username: str*, *member_fullname: str*, *date: datetime.datetime*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.trello.`**`MoveCardEvent`**(*old_list_id: str*, *old_list_name: str*, **args*, ***kwargs*)
Event triggered when a card is moved to another list.

`__init__`(*old_list_id: str*, *old_list_name: str*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.trello.`**`NewCardEvent`**(*card_id: str*, *card_name: str*, *list_id: str*, *list_name: str*, *board_id: str*, *board_name: str*, *closed: bool*, *member_id: str*, *member_username: str*, *member_fullname: str*, *date: datetime.datetime*, **args*, ***kwargs*)
Event triggered when a card is created.

class `platypush.message.event.trello.`**`TrelloEvent`**(*target=None*, *origin=None*, *id=None*, *timestamp=None*, *disable_logging=False*, *disable_web_clients_notification=False*, ***kwargs*)

class `platypush.message.event.trello.`**`UnarchivedCardEvent`**(**args*, ***kwargs*)
Event triggered when a card is un-archived.

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.53 `platypush.message.event.video`

class `platypush.message.event.video.`**`NewPlayingVideoEvent`**(*video=None*, **args*, ***kwargs*)
Event triggered when a video playback is paused

__init__(*video=None*, **args*, ***kwargs*)

Parameters **video** (str) – File name or URI of the played video

class platypush.message.event.video.**VideoEvent**(**args*, ***kwargs*)
Base class for video events

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.video.**VideoPauseEvent**(**args*, ***kwargs*)
Event triggered when a video playback is paused

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.video.**VideoPlayEvent**(*video=None*, **args*, ***kwargs*)
Event triggered when a new video content is played

__init__(*video=None*, **args*, ***kwargs*)

Parameters **video** (str) – File name or URI of the played video

class platypush.message.event.video.**VideoStopEvent**(**args*, ***kwargs*)
Event triggered when a video is stopped

__init__(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.54 platypush.message.event.weather

class platypush.message.event.weather.**NewPrecipitationForecastEvent**(**args*, *plugin_name: Optional[str] = None*, *average: float*, *total: float*, *time_frame: int*, ***kwargs*)
Event triggered when the precipitation forecast changes

__init__(**args*, *plugin_name: Optional[str] = None*, *average: float*, *total: float*, *time_frame: int*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.weather.**NewWeatherConditionEvent**(**args*, *plugin_name: Optional[str] = None*, ***kwargs*)

Event triggered when the weather condition changes

`__init__`(**args*, *plugin_name: Optional[str] = None*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.55 `platypush.message.event.web`

class `platypush.message.event.web.`**`DashboardIframeUpdateEvent`**(*url*, *width=None*, *height=None*, *timeout=None*, **args*, ***kwargs*)
Deliver a DashboardIframeUpdateEvent if you are using the web dashboard and you want the connected web clients to show a certain URL in the iframe modal window for (optionally) a certain time.

`__init__`(*url*, *width=None*, *height=None*, *timeout=None*, **args*, ***kwargs*)

Parameters

- **`url`** (*`str`*) – URL to show in the iframe dashboard element
- **`width`** – Iframe width, as int (pixels) or CSS string
- **`height`** – Iframe height, as int (pixels) or CSS string
- **`timeout`** (*`float`*) – If set, the iframe will be closed after this time (in seconds)

3.56 `platypush.message.event.web.widget`

class `platypush.message.event.web.widget.`**`WidgetUpdateEvent`**(*widget*, **args*, ***kwargs*)
Event delivered to the dashboard when a widget update request is delivered to it

`__init__`(*widget*, **args*, ***kwargs*)

Parameters **`widget`** (*`str`*) – Widget ID

3.57 `platypush.message.event.wiimote`

class `platypush.message.event.wiimote.`**`WiimoteConnectionEvent`**(**args*, ***kwargs*)
Event triggered upon Wiimote connection

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.wiimote.`**`WiimoteDisconnectionEvent`**(**args*, ***kwargs*)
Event triggered upon Wiimote disconnection

`__init__`(**args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.wiimote.`**`WiimoteEvent`**(**args*, ***kwargs*)
Event triggered upon Wiimote event

__init__(**args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.58 platypush.message.event.zeroborg

class platypush.message.event.zeroborg.**ZeroborgDriveEvent**(*motors: Union[list, tuple], direction: Optional[str] = None, *args, **kwargs*)

__init__(*motors: Union[list, tuple], direction: Optional[str] = None, *args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zeroborg.**ZeroborgEvent**(*target=None, origin=None, id=None, timestamp=None, disable_logging=False, disable_web_clients_notification=False, **kwargs*)

class platypush.message.event.zeroborg.**ZeroborgStopEvent**(*target=None, origin=None, id=None, timestamp=None, disable_logging=False, disable_web_clients_notification=False, **kwargs*)

3.59 platypush.message.event.zeroconf

class platypush.message.event.zeroconf.**ZeroconfEvent**(*service_event: platypush.message.event.zeroconf.ZeroconfEventType, service_type: str, service_name: str, service_info: Optional[dict] = None, *args, **kwargs*)

__init__(*service_event: platypush.message.event.zeroconf.ZeroconfEventType, service_type: str, service_name: str, service_info: Optional[dict] = None, *args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zeroconf.**ZeroconfEventType**
An enumeration.

class platypush.message.event.zeroconf.**ZeroconfServiceAddedEvent**(**args, **kwargs*)
Event triggered when a service is added or discovered.

__init__(**args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.zeroconf.`**`ZeroconfServiceRemovedEvent`**(**args*, ***kwargs*)

Event triggered when a service is removed.

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.zeroconf.`**`ZeroconfServiceUpdatedEvent`**(**args*, ***kwargs*)

Event triggered when a service is updated.

`__init__`(**args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.60 `platypush.message.event.zigbee.mqtt`

class `platypush.message.event.zigbee.mqtt.`**`ZigbeeMqttDeviceBannedEvent`**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when a device is banned from the network.

`__init__`(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.zigbee.mqtt.`**`ZigbeeMqttDeviceBindEvent`**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when a device bind occurs on the network.

`__init__`(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class `platypush.message.event.zigbee.mqtt.`**`ZigbeeMqttDeviceConnectedEvent`**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when a device connects to the network.

`__init__`(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttDevicePairingEvent**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when a device is pairing to the network.

__init__(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttDevicePropertySetEvent**(*host: str, port: int, device: str, properties: Dict[str, Any], *args, **kwargs*)

Triggered when a the properties of a Zigbee connected devices (state, brightness, alert etc.) change.

__init__(*host: str, port: int, device: str, properties: Dict[str, Any], *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttDeviceRemovedEvent**(*host: str*, *port: int*, *device=None*, *force=False*, **args*, ***kwargs*)

Triggered when a device is removed from the network.

__init__(*host: str*, *port: int*, *device=None*, *force=False*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttDeviceRemovedFailedEvent**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when the removal of a device from the network failed.

__init__(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttDeviceRenamedEvent**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when a device is renamed on the network.

__init__(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttDeviceUnbindEvent**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when a device bind occurs on the network.

__init__(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttDeviceWhitelistedEvent**(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)

Triggered when a device is whitelisted on the network.

__init__(*host: str*, *port: int*, *device=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttErrorEvent**(*host: str*, *port: int*, *error=None*, **args*, ***kwargs*)

Triggered when an error happens on the zigbee2mqtt service.

__init__(*host: str*, *port: int*, *error=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttEvent**(*target=None*, *origin=None*, *id=None*, *timestamp=None*, *disable_logging=False*, *disable_web_clients_notification=False*, ***kwargs*)

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttGroupAddedEvent**(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)

Triggered when a group is added.

__init__(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttGroupAddedFailedEvent**(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)

Triggered when a request to add a group fails.

__init__(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttGroupRemoveAllEvent**(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)

Triggered when all the devices are removed from a group.

__init__(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttGroupRemoveAllFailedEvent**(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)

Triggered when a request to remove all the devices from a group fails.

__init__(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttGroupRemovedEvent**(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)

Triggered when a group is removed.

__init__(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttGroupRemovedFailedEvent**(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)

Triggered when a request to remove a group fails.

__init__(*host: str*, *port: int*, *group=None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttOfflineEvent**(*host: str*, *port: int*, **args*, ***kwargs*)

Triggered when a zigbee2mqtt service goes offline.

__init__(*host: str*, *port: int*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zigbee.mqtt.**ZigbeeMqttOnlineEvent**(*host: str*, *port: int*, **args*, ***kwargs*)

Triggered when a zigbee2mqtt service goes online.

__init__(*host: str*, *port: int*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

3.61 platypush.message.event.zwave

class platypush.message.event.zwave.**ZwaveButtonCreatedEvent**(*node: Dict[str, Any], *args, **kwargs*)

Triggered when a button is added to the network.

class platypush.message.event.zwave.**ZwaveButtonOffEvent**(*node: Dict[str, Any], *args, **kwargs*)

Triggered when a button is released.

class platypush.message.event.zwave.**ZwaveButtonOnEvent**(*node: Dict[str, Any], *args, **kwargs*)

Triggered when a button is pressed.

class platypush.message.event.zwave.**ZwaveButtonRemovedEvent**(*node: Dict[str, Any], *args, **kwargs*)

Triggered when a button is removed from the network.

class platypush.message.event.zwave.**ZwaveCommandEvent**(*state: str, state_description: str, error: Optional[str] = None, error_description: Optional[str] = None, node: Optional[Dict[str, Any]] = None, *args, **kwargs*)

Triggered when a command is received on the network.

__init__(*state: str, state_description: str, error: Optional[str] = None, error_description: Optional[str] = None, node: Optional[Dict[str, Any]] = None, *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zwave.**ZwaveCommandWaitingEvent**(*state: str, state_description: str, error: Optional[str] = None, error_description: Optional[str] = None, node: Optional[Dict[str, Any]] = None, *args, **kwargs*)

Triggered when a command is waiting for a message to proceed.

class platypush.message.event.zwave.**ZwaveEvent**(*device: Optional[str] = None, *args, **kwargs*)

__init__(*device: Optional[str] = None, *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zwave.**ZwaveNetworkErrorEvent**(*device: Optional[str] = None, *args, **kwargs*)

Triggered when an error occurs on the Z-Wave network.

class platypush.message.event.zwave.**ZwaveNetworkReadyEvent**(*ozw_library_version: str, python_library_version: str, zwave_library: str, node_id: int, node_version: str, home_id: int, nodes_count: int, device: Optional[str] = None, *args, **kwargs*)

Triggered when the network started on a Z-Wave adapter becomes ready.

__init__(*ozw_library_version: str, python_library_version: str, zwave_library: str, node_id: int, node_version: str, home_id: int, nodes_count: int, device: Optional[str] = None, *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zwave.**ZwaveNetworkResetEvent**(*device: Optional[str] = None*, **args*, ***kwargs*)
Triggered when a Z-Wave network is reset.

class platypush.message.event.zwave.**ZwaveNetworkStoppedEvent**(*device: Optional[str] = None*, **args*, ***kwargs*)
Triggered when a Z-Wave network is stopped.

class platypush.message.event.zwave.**ZwaveNodeAddedEvent**(*node: Dict[str, Any], *args, **kwargs*)
Triggered when a node is added to the network.

class platypush.message.event.zwave.**ZwaveNodeEvent**(*node: Dict[str, Any], *args, **kwargs*)
Generic Z-Wave node event class.

__init__(*node: Dict[str, Any], *args, **kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zwave.**ZwaveNodeGroupEvent**(*group_index: Optional[int] = None*, **args*, ***kwargs*)
Triggered when a node is associated/de-associated to a group.

__init__(*group_index: Optional[int] = None*, **args*, ***kwargs*)
Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zwave.**ZwaveNodePollingDisabledEvent**(*node: Dict[str, Any], *args, **kwargs*)
Triggered when the polling of a node is successfully turned off.

class platypush.message.event.zwave.**ZwaveNodePollingEnabledEvent**(*node: Dict[str, Any], *args, **kwargs*)
Triggered when the polling of a node is successfully turned on.

class platypush.message.event.zwave.**ZwaveNodeQueryCompletedEvent**(*device: Optional[str] = None*, **args*, ***kwargs*)
Triggered when all the nodes on the network have been queried.

class platypush.message.event.zwave.**ZwaveNodeReadyEvent**(*node: Dict[str, Any], *args, **kwargs*)
Triggered when a node is ready.

class platypush.message.event.zwave.**ZwaveNodeRemovedEvent**(*node: Dict[str, Any], *args, **kwargs*)
Triggered when a node is removed from the network.

class platypush.message.event.zwave.**ZwaveNodeRenamedEvent**(*node: Dict[str, Any], *args, **kwargs*)
Triggered when a node is renamed.

class platypush.message.event.zwave.**ZwaveNodeSceneEvent**(*scene_id: int*, **args*, ***kwargs*)

Triggered when a scene is activated on a node.

__init__(*scene_id: int*, **args*, ***kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zwave.**ZwaveValueAddedEvent**(*node: Dict[str, Any], value: Dict[str, Any], *args, **kwargs*)

Triggered when a value is added to a node on the network.

class platypush.message.event.zwave.**ZwaveValueChangedEvent**(*node: Dict[str, Any], value: Dict[str, Any], *args, **kwargs*)

Triggered when a value of a node on the network changes.

class platypush.message.event.zwave.**ZwaveValueEvent**(*node: Dict[str, Any], value: Dict[str, Any], *args, **kwargs*)

Abstract class for Z-Wave value events.

__init__(*node: Dict[str, Any], value: Dict[str, Any], *args, **kwargs*)

Params: target – Target node [String] origin – Origin node (default: current node) [String] id – Event ID (default: auto-generated) kwargs – Additional arguments for the event [kwDict]

class platypush.message.event.zwave.**ZwaveValueRefreshedEvent**(*node: Dict[str, Any], value: Dict[str, Any], *args, **kwargs*)

Triggered when a value of a node on the network is refreshed.

class platypush.message.event.zwave.**ZwaveValueRemovedEvent**(*node: Dict[str, Any], value: Dict[str, Any], *args, **kwargs*)

Triggered when a value of a node on the network is removed.

CHAPTER 4

Responses

4.1 `platypush.message.response.bluetooth`

class `platypush.message.response.bluetooth.`**`BluetoothDiscoverCharacteristicsResponse`**(*character list*, **args*, ***kwargs*

Example services response output:

```
[
  {
    "uuid": "00002a00-0000-1000-8000-00805f9b34fb",
    "handle": 2,
    "properties": 10,
    "value_handle": 3
  },
  {
    "uuid": "00002a01-0000-1000-8000-00805f9b34fb",
    "handle": 4,
    "properties": 2,
    "value_handle": 5
  },
  {
    "uuid": "00002a04-0000-1000-8000-00805f9b34fb",
    "handle": 6,
    "properties": 2,
    "value_handle": 7
  },
  {
    "uuid": "0000fec8-0000-1000-8000-00805f9b34fb",
    "handle": 10,
    "properties": 32,
    "value_handle": 11
  },
```

(continues on next page)

(continued from previous page)

```
  {
    "uuid": "0000fec7-0000-1000-8000-00805f9b34fb",
    "handle": 13,
    "properties": 8,
    "value_handle": 14
  },
  {
    "uuid": "0000fec9-0000-1000-8000-00805f9b34fb",
    "handle": 15,
    "properties": 2,
    "value_handle": 16
  },
  {
    "uuid": "cba20003-224d-11e6-9fb8-0002a5d5c51b",
    "handle": 18,
    "properties": 16,
    "value_handle": 19
  },
  {
    "uuid": "cba20002-224d-11e6-9fb8-0002a5d5c51b",
    "handle": 21,
    "properties": 12,
    "value_handle": 22
  }
]
```

`__init__`(*characteristics: list*, **args*, ***kwargs*)

Parameters

- **`target`** (*`str`*) – Target
- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to
- **`timestamp`** (*`float`*) – Message timestamp

class `platypush.message.response.bluetooth.`**`BluetoothDiscoverPrimaryResponse`**(*services: list*, **args*, ***kwargs*)

Example services response output:

```
[
  {
    "uuid": "00001800-0000-1000-8000-00805f9b34fb",
    "start": 1,
    "end": 7
  },
  {
    "uuid": "00001801-0000-1000-8000-00805f9b34fb",
    "start": 8,
    "end": 8
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "uuid": "0000fee7-0000-1000-8000-00805f9b34fb",
    "start": 9,
    "end": 16
  },
  {
    "uuid": "cba20d00-224d-11e6-9fb8-0002a5d5c51b",
    "start": 17,
    "end": 65535
  }
]
```

__init__(*services: list*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.bluetooth.**BluetoothLookupNameResponse**(*addr: str*, *name: str*, **args*, ***kwargs*)

__init__(*addr: str*, *name: str*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.bluetooth.**BluetoothLookupServiceResponse**(*services: list*, **args*, ***kwargs*)

Example services response output:

```
[
  {
    "service-classes": [
      "1801"
    ],
    "profiles": [],
```

(continues on next page)

(continued from previous page)

```
    "name": "Service name #1",
    "description": null,
    "provider": null,
    "service-id": null,
    "protocol": "L2CAP",
    "port": 31,
    "host": "00:11:22:33:44:55"
  },
  {
    "service-classes": [
      "1800"
    ],
    "profiles": [],
    "name": "Service name #2",
    "description": null,
    "provider": null,
    "service-id": null,
    "protocol": "L2CAP",
    "port": 31,
    "host": "00:11:22:33:44:56"
  },
  {
    "service-classes": [
      "1112",
      "1203"
    ],
    "profiles": [
      [
        "1108",
        258
      ]
    ],
    "name": "Headset Gateway",
    "description": null,
    "provider": null,
    "service-id": null,
    "protocol": "RFCOMM",
    "port": 2,
    "host": "00:11:22:33:44:57"
  }
]
```

__init__(*services: list*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.bluetooth.**BluetoothResponse**(*target=None*, *origin=None*, *id=None*, *output=None*, *errors=None*, *timestamp=None*, *disable_logging=False*)

class platypush.message.response.bluetooth.**BluetoothScanResponse**(*devices*, **args*, ***kwargs*)

__init__(*devices*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.2 platypush.message.response.camera

class platypush.message.response.camera.**CameraResponse**(*target=None*, *origin=None*, *id=None*, *output=None*, *errors=None*, *timestamp=None*, *disable_logging=False*)

4.3 platypush.message.response.camera.android

class platypush.message.response.camera.android.**AndroidCameraPictureResponse**(*image_file: str*, **args*, ***kwargs*)

__init__(*image_file: str*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.camera.android.**AndroidCameraStatusListResponse**(*status: List[platypush.r* ***kwargs*)

__init__(*status: List[platypush.message.response.camera.android.AndroidCameraStatusResponse], **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.camera.android.**AndroidCameraStatusResponse**(**args, name: str = None, stream_url: str = None, image_url: str = None, audio_url: str = None, orientation: str = None, idle: str = None, audio_only: str = None, overlay: str = None, quality: str = None, focus_homing: str = None, ip_address: str = None, motion_limit: str = None, adet_limit:*

Example response:

```
{
    "stream_url": "https://192.168.1.30:8080/video",
    "image_url": "https://192.168.1.30:8080/photo.jpg",
    "audio_url": "https://192.168.1.30:8080/audio.wav",
    "orientation": "landscape",
    "idle": "off",
    "audio_only": "off",
    "overlay": "off",
    "quality": "49",
    "focus_homing": "off",
    "ip_address": "192.168.1.30",
    "motion_limit": "250",
    "adet_limit": "200",
    "night_vision": "off",
    "night_vision_average": "2",
    "night_vision_gain": "1.0",
    "motion_detect": "off",
    "motion_display": "off",
    "video_chunk_len": "60",
    "gps_active": "off",
    "video_size": "1920x1080",
    "mirror_flip": "none",
    "ffc": "off",
    "rtsp_video_formats": "",
    "rtsp_audio_formats": "",
    "video_connections": "0",
    "audio_connections": "0",
    "ivideon_streaming": "off",
    "zoom": "100",
    "crop_x": "50",
    "crop_y": "50",
    "coloreffect": "none",
    "scenemode": "auto",
    "focusmode": "continuous-video",
    "whitebalance": "auto",
    "flashmode": "off",
    "antibanding": "off",
    "torch": "off",
    "focus_distance": "0.0",
    "focal_length": "4.25",
    "aperture": "1.7",
    "filter_density": "0.0",
    "exposure_ns": "9384",
    "frame_duration": "33333333",
    "iso": "100",
    "manual_sensor": "off",
    "photo_size": "1920x1080"
}
```

__init__(**args, name: str = None, stream_url: str = None, image_url: str = None, audio_url: str = None, orientation: str = None, idle: str = None, audio_only: str = None, overlay: str = None, quality: str = None, focus_homing: str = None, ip_address: str = None, motion_limit: str = None, adet_limit: str = None, night_vision: str = None, night_vision_average: str = None, night_vision_gain: str = None, motion_detect: str = None, motion_display: str = None, video_chunk_len: str = None, gps_active: str = None, video_size: str = None, mirror_flip: str = None, ffc: str = None, rtsp_video_formats: str = None, rtsp_audio_formats: str = None, video_connections: str = None, audio_connections: str = None, ivideon_streaming: str = None, zoom: str = None, crop_x: str = None, crop_y: str = None, coloreffect: str = None, scenemode: str = None, focusmode: str = None, whitebalance: str = None, flashmode: str = None, antibanding: str = None, torch: str = None, focus_distance: str = None, focal_length: str = None, aperture: str = None, filter_density: str = None, exposure_ns: str = None, frame_duration: str = None, iso: str = None, manual_sensor: str = None, photo_size: str = None, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.4 platypush.message.response.chat.telegram

class platypush.message.response.chat.telegram.**TelegramChatResponse**(*chat_id: int, link: str, username: str, invite_link: Optional[str], title: Optional[str] = None, description: Optional[str] = None, type: Optional[str] = None, first_name: Optional[str] = None, last_name: Optional[str] = None, *args, **kwargs*)

__init__(*chat_id: int, link: str, username: str, invite_link: Optional[str], title: Optional[str] = None, description: Optional[str] = None, type: Optional[str] = None, first_name: Optional[str] = None, last_name: Optional[str] = None, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.chat.telegram.**TelegramFileResponse**(*file_id: str*, *file_path: str*, *file_size: int*, **args*, ***kwargs*)

__init__(*file_id: str*, *file_path: str*, *file_size: int*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.chat.telegram.**TelegramMessageResponse**(*message_id: int, chat_id: int, creation_date: Optional[datetime.datetime], chat_username: Optional[str] = None, chat_firstname: Optional[str] = None, chat_lastname: Optional[str] = None, from_user_id: Optional[int] = None, from_username: Optional[str] = None, from_firstname: Optional[str] = None, from_lastname: Optional[str] = None, text: Optional[str] = None, caption: Optional[str] = None, edit_date: Optional[datetime.datetime] = None, for-*

__init__(*message_id: int, chat_id: int, creation_date: Optional[datetime.datetime], chat_username: Optional[str] = None, chat_firstname: Optional[str] = None, chat_lastname: Optional[str] = None, from_user_id: Optional[int] = None, from_username: Optional[str] = None, from_firstname: Optional[str] = None, from_lastname: Optional[str] = None, text: Optional[str] = None, caption: Optional[str] = None, edit_date: Optional[datetime.datetime] = None, forward_date: Optional[datetime.datetime] = None, forward_from_message_id: Optional[int] = None, photo_file_id: Optional[str] = None, photo_file_size: Optional[int] = None, photo_width: Optional[int] = None, photo_height: Optional[int] = None, document_file_id: Optional[str] = None, document_file_name: Optional[str] = None, document_file_size: Optional[str] = None, document_mime_type: Optional[str] = None, audio_file_id: Optional[str] = None, audio_file_size: Optional[str] = None, audio_mime_type: Optional[str] = None, audio_performer: Optional[str] = None, audio_title: Optional[str] = None, audio_duration: Optional[str] = None, location_latitude: Optional[float] = None, location_longitude: Optional[float] = None, contact_phone_number: Optional[str] = None, contact_first_name: Optional[str] = None, contact_last_name: Optional[str] = None, contact_user_id: Optional[int] = None, contact_vcard: Optional[str] = None, video_file_id: Optional[str] = None, video_file_size: Optional[int] = None, video_width: Optional[int] = None, video_height: Optional[int] = None, video_mime_type: Optional[str] = None, video_duration: Optional[str] = None, link: Optional[str] = None, media_group_id: Optional[int] = None, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.chat.telegram.**TelegramUserResponse**(*user_id: int, username: str, is_bot: bool, first_name: str, last_name: Optional[str] = None, language_code: Optional[str] = None, link: Optional[str] = None, *args, **kwargs*)

__init__(*user_id: int*, *username: str*, *is_bot: bool*, *first_name: str*, *last_name: Optional[str] = None*, *language_code: Optional[str] = None*, *link: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.chat.telegram.**TelegramUsersResponse**(*users: List[platypush.message.respons*
**args*, ***kwargs*)

__init__(*users: List[platypush.message.response.chat.telegram.TelegramUserResponse]*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.5 platypush.message.response.google.drive

class platypush.message.response.google.drive.**GoogleDriveFile**(*type: str*, *id: str*, *name: str*, *mime_type: Optional[str] = None*, **args*, ***kwargs*)

__init__(*type: str*, *id: str*, *name: str*, *mime_type: Optional[str] = None*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.google.drive.**GoogleDriveResponse**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

4.6 platypush.message.response.linode

class platypush.message.response.linode.**LinodeBackupModel**(*backup: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392054110>*)

__init__(*backup: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392054110>*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.linode.**LinodeConfigModel**(*config: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392541a90>*)

__init__(*config: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392541a90>*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.linode.**LinodeDiskModel**(*disk: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392541390>*)

__init__(*disk: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392541390>*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.linode.**LinodeImageModel**(*image: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc3920546d0>*)

__init__(*image: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc3920546d0>*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.linode.**LinodeInstanceModel**(*node: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391cd8750>*)

__init__(*node: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391cd8750>*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.linode.**LinodeInstanceResponse**(*instance: <sphinx.ext.autodoc.importer._MockObj object at 0x7fc391cd8750>, *args, **kwargs*)

__init__(*instance: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391cd8750>, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.linode.**LinodeInstancesResponse**(*instances: List[<sphinx.ext.autodoc.importer._Mo object at 0x7fc391cd8750>], *args, **kwargs*)

__init__(*instances: List[<sphinx.ext.autodoc.importer._MockObject object at 0x7fc391cd8750>], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.linode.**LinodeKernelModel**(*kernel: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391d92590>*)

__init__(*kernel: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391d92590>*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.linode.**LinodeResponse**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

class platypush.message.response.linode.**LinodeTypeModel**(*type: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391d92e90>*)

__init__(*type: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391d92e90>*)
Initialize self. See help(type(self)) for accurate signature.

4.7 platypush.message.response.pihole

class platypush.message.response.pihole.**PiholeStatusResponse**(*server: str, status: str, ads_percentage: float, blocked: int, cached: int, domain_count: int, forwarded: int, queries: int, total_clients: int, total_queries: int, unique_clients: int, unique_domains: int, version: str, *args, **kwargs*)

__init__(*server: str, status: str, ads_percentage: float, blocked: int, cached: int, domain_count: int, forwarded: int, queries: int, total_clients: int, total_queries: int, unique_clients: int, unique_domains: int, version: str, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.8 platypush.message.response.ping

class platypush.message.response.ping.**PingResponse**(*host: str, success: bool, *args, min: Optional[float] = None, max: Optional[float] = None, avg: Optional[float] = None, mdev: Optional[float] = None, **kwargs*)

__init__(*host: str, success: bool, *args, min: Optional[float] = None, max: Optional[float] = None, avg: Optional[float] = None, mdev: Optional[float] = None, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.9 platypush.message.response.printer.cups

class platypush.message.response.printer.cups.**PrinterJobAddedResponse**(**args, printer: str, job_id: int, **kwargs*)

__init__(**args, printer: str, job_id: int, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.printer.cups.**PrinterResponse**(**args, name: str, printer_type: int, info: str, uri: str, state: int, is_shared: bool, state_message: Optional[str] = None, state_reasons: Optional[List[str]] = None, location: Optional[str] = None, uri_supported: Optional[str] = None, make_and_model: Optional[str] = None, **kwargs*)

__init__(**args, name: str, printer_type: int, info: str, uri: str, state: int, is_shared: bool, state_message: Optional[str] = None, state_reasons: Optional[List[str]] = None, location: Optional[str] = None, uri_supported: Optional[str] = None, make_and_model: Optional[str] = None, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.printer.cups.**PrintersResponse**(**args, printers: List[platypush.message.response.printer.c **kwargs*)

__init__(**args, printers: List[platypush.message.response.printer.cups.PrinterResponse], **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.10 platypush.message.response.qrcode

class platypush.message.response.qrcode.**QrcodeDecodedResponse**(*results: List[<sphinx.ext.autodoc.importer._MockO object at 0x7fc392081f90>], image_file: Optional[str] = None, *args, **kwargs*)

__init__(*results: List[<sphinx.ext.autodoc.importer._MockObject object at 0x7fc392081f90>], image_file: Optional[str] = None, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors

- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.qrcode.**QrcodeGeneratedResponse**(*content: str, format: str, data: Optional[str] = None, image_file: Optional[str] = None, *args, **kwargs*)

__init__(*content: str, format: str, data: Optional[str] = None, image_file: Optional[str] = None, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.qrcode.**QrcodeResponse**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

class platypush.message.response.qrcode.**RectModel**(*rect: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392081c50>*)

__init__(*rect: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392081c50>*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.qrcode.**ResultModel**(*result: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392081f90>, *args, **kwargs*)

__init__(*result: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc392081f90>, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

4.11 platypush.message.response.ssh

class platypush.message.response.ssh.**SSHKeygenResponse**(*fingerprint: str, key_file: str, pub_key_file: str, *args, **kwargs*)

__init__(*fingerprint: str, key_file: str, pub_key_file: str, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.ssh.**SSHResponse**(**args*, ***kwargs*)

__init__(**args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.12 platypush.message.response.stt

class platypush.message.response.stt.**SpeechDetectedResponse**(**args*, *speech: str*, ***kwargs*)

__init__(**args*, *speech: str*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.13 platypush.message.response.system

class platypush.message.response.system.**ConnectUserResponse**(*name: str*, *terminal: str*, *host: str*, *started: datetime.datetime*, *pid: Optional[int] = None*, **args*, ***kwargs*)

__init__(*name: str*, *terminal: str*, *host: str*, *started: datetime.datetime*, *pid: Optional[int] = None*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**ConnectedUserResponseList**(*responses: List[platypush.message.response.sys* **args*, ***kwargs*)

__init__(*responses: List[platypush.message.response.system.ConnectUserResponse]*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**CpuFrequencyResponse**(*min: int*, *max: int*, *current: int*, **args*, ***kwargs*)

__init__(*min: int*, *max: int*, *current: int*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**CpuInfoResponse**(*arch: str, bits: int, count: int, vendor_id: str, brand: str, hz_advertised: int, hz_actual: int, model: int, flags: List[str], family: Optional[int], stepping: Optional[int], l1_instruction_cache_size: Union[int, str, None], l1_data_cache_size: Union[int, str, None], l2_cache_size: Union[int, str, None], l3_cache_size: Union[int, str, None], *args, **kwargs*)

__init__(*arch: str, bits: int, count: int, vendor_id: str, brand: str, hz_advertised: int, hz_actual: int, model: int, flags: List[str], family: Optional[int], stepping: Optional[int], l1_instruction_cache_size: Union[int, str, None], l1_data_cache_size: Union[int, str, None], l2_cache_size: Union[int, str, None], l3_cache_size: Union[int, str, None], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**CpuResponse**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

class platypush.message.response.system.**CpuResponseList**(*responses: List[platypush.message.response.system.CpuRespor *args, **kwargs*)

__init__(*responses: List[platypush.message.response.system.CpuResponse], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**CpuStatsResponse**(*ctx_switches: int*, *interrupts: int*, *soft_interrupts: int*, *syscalls: int*, **args*, ***kwargs*)

__init__(*ctx_switches: int*, *interrupts: int*, *soft_interrupts: int*, *syscalls: int*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**CpuTimesResponse**(*user: float*, *nice: float*, *system: float*, *idle: float*, *iowait: float*, *irq: float*, *softirq: float*, *steal: float*, *guest: float*, *guest_nice: float*, **args*, ***kwargs*)

__init__(*user: float*, *nice: float*, *system: float*, *idle: float*, *iowait: float*, *irq: float*, *softirq: float*, *steal: float*, *guest: float*, *guest_nice: float*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**DiskIoCountersResponse**(*read_count: int*, *write_count: int*, *read_bytes: int*, *write_bytes: int*, *read_time: int*, *write_time: int*, *read_merged_count: int*, *write_merged_count: int*, *busy_time: int*, *disk: Optional[str] = None*, **args*, ***kwargs*)

__init__(*read_count: int*, *write_count: int*, *read_bytes: int*, *write_bytes: int*, *read_time: int*, *write_time: int*, *read_merged_count: int*, *write_merged_count: int*, *busy_time: int*, *disk: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**DiskPartitionResponse**(*device: str*, *mount_point: str*, *fstype: Optional[str] = None*, *opts: Optional[str] = None*, **args*, ***kwargs*)

__init__(*device: str*, *mount_point: str*, *fstype: Optional[str] = None*, *opts: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**DiskResponse**(*target=None*, *origin=None*, *id=None*, *output=None*, *errors=None*, *timestamp=None*, *disable_logging=False*)

class platypush.message.response.system.**DiskResponseList**(*responses: List[platypush.message.response.system.DiskResp*
**args, **kwargs*)

__init__(*responses: List[platypush.message.response.system.DiskResponse], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to

- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**DiskUsageResponse**(*path: str*, *total: int*, *used: int*, *free: int*, *percent: float*, **args*, ***kwargs*)

__init__(*path: str*, *total: int*, *used: int*, *free: int*, *percent: float*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**MemoryResponse**(*target=None*, *origin=None*, *id=None*, *output=None*, *errors=None*, *timestamp=None*, *disable_logging=False*)

class platypush.message.response.system.**NetworkAddressResponse**(*nic: str*, *ipv4_address: Optional[str] = None*, *ipv4_netmask: Optional[str] = None*, *ipv4_broadcast: Optional[str] = None*, *ipv6_address: Optional[str] = None*, *ipv6_netmask: Optional[str] = None*, *ipv6_broadcast: Optional[str] = None*, *mac_address: Optional[str] = None*, *mac_broadcast: Optional[str] = None*, *ptp: Optional[str] = None*, **args*, ***kwargs*)

__init__(*nic: str*, *ipv4_address: Optional[str] = None*, *ipv4_netmask: Optional[str] = None*, *ipv4_broadcast: Optional[str] = None*, *ipv6_address: Optional[str] = None*, *ipv6_netmask: Optional[str] = None*, *ipv6_broadcast: Optional[str] = None*, *mac_address: Optional[str] = None*, *mac_broadcast: Optional[str] = None*, *ptp: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**NetworkConnectionResponse**(*fd: int*, *family: str*, *type: str*, *local_address: str*, *local_port: int*, *remote_address: str*, *remote_port: int*, *status: str*, *pid: int*, **args*, ***kwargs*)

__init__(*fd: int*, *family: str*, *type: str*, *local_address: str*, *local_port: int*, *remote_address: str*, *remote_port: int*, *status: str*, *pid: int*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**NetworkInterfaceStatsResponse**(*nic: str*, *is_up: bool*, *duplex: str*, *speed: int*, *mtu: int*, **args*, ***kwargs*)

__init__(*nic: str*, *is_up: bool*, *duplex: str*, *speed: int*, *mtu: int*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**NetworkIoCountersResponse**(*bytes_sent: int*, *bytes_recv: int*, *packets_sent: int*, *packets_recv: int*, *errin: int*, *errout: int*, *dropin: int*, *dropout: int*, *nic: Optional[str] = None*, **args*, ***kwargs*)

__init__(*bytes_sent: int*, *bytes_recv: int*, *packets_sent: int*, *packets_recv: int*, *errin: int*, *errout: int*, *dropin: int*, *dropout: int*, *nic: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors

- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**NetworkResponse**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

class platypush.message.response.system.**NetworkResponseList**(*responses: List[platypush.message.response.system.Netw *args, **kwargs*)

__init__(*responses: List[platypush.message.response.system.NetworkResponse], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**ProcessResponse**(*pid: int, name: str, started: datetime.datetime, ppid: Optional[int], children: Optional[List[int]] = None, exe: Optional[List[str]] = None, status: Optional[str] = None, username: Optional[str] = None, terminal: Optional[str] = None, cpu_user_time: Optional[float] = None, cpu_system_time: Optional[float] = None, cpu_children_user_time: Optional[float] = None, cpu_children_system_time: Optional[float] = None, mem_rss: Optional[int] = None, mem_vms: Optional[int] = None, mem_shared: Optional[int] = None, mem_text: Optional[int] = None, mem_data: Optional[int] = None, mem_lib: Optional[int] = None, mem_dirty: Optional[int] = None, mem_percent: Optional[float] = None, *args, **kwargs*)

__init__(*pid: int, name: str, started: datetime.datetime, ppid: Optional[int], children: Optional[List[int]] = None, exe: Optional[List[str]] = None, status: Optional[str] = None, username: Optional[str] = None, terminal: Optional[str] = None, cpu_user_time: Optional[float] = None, cpu_system_time: Optional[float] = None, cpu_children_user_time: Optional[float] = None, cpu_children_system_time: Optional[float] = None, mem_rss: Optional[int] = None, mem_vms: Optional[int] = None, mem_shared: Optional[int] = None, mem_text: Optional[int] = None, mem_data: Optional[int] = None, mem_lib: Optional[int] = None, mem_dirty: Optional[int] = None, mem_percent: Optional[float] = None, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**ProcessResponseList**(*responses: List[platypush.message.response.system.Proc *args, **kwargs*)

__init__(*responses: List[platypush.message.response.system.ProcessResponse], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**SensorBatteryResponse**(*percent: float, secs_left: int, power_plugged: bool, *args, **kwargs*)

__init__(*percent: float, secs_left: int, power_plugged: bool, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**SensorFanResponse**(*name: str, current: int, label: Optional[str] = None, *args, **kwargs*)

__init__(*name: str, current: int, label: Optional[str] = None, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**SensorResponse**(*target=None*, *origin=None*, *id=None*, *output=None*, *errors=None*, *timestamp=None*, *disable_logging=False*)

class platypush.message.response.system.**SensorResponseList**(*responses: List[platypush.message.response.system.Senso* **args, **kwargs*)

__init__(*responses: List[platypush.message.response.system.SensorResponse], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**SensorTemperatureResponse**(*name: str*, *current: float*, *high: Optional[float] = None*, *critical: Optional[float] = None*, *label: Optional[str] = None*, **args*, ***kwargs*)

__init__(*name: str*, *current: float*, *high: Optional[float] = None*, *critical: Optional[float] = None*, *label: Optional[str] = None*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**SwapMemoryUsageResponse**(*total: int, percent: float, used: int, free: int, sin: int, sout: int, *args, **kwargs*)

__init__(*total: int, percent: float, used: int, free: int, sin: int, sout: int, *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.system.**SystemResponse**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

class platypush.message.response.system.**SystemResponseList**(*responses: List[platypush.message.response.system.Systen *args, **kwargs*)

__init__(*responses: List[platypush.message.response.system.SystemResponse], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class `platypush.message.response.system.`**`VirtualMemoryUsageResponse`**(*total: int, available: int, percent: float, used: int, free: int, active: int, inactive: int, buffers: int, cached: int, shared: int, *args, **kwargs*)

`__init__`(*total: int, available: int, percent: float, used: int, free: int, active: int, inactive: int, buffers: int, cached: int, shared: int, *args, **kwargs*)

Parameters

- **`target`** (*`str`*) – Target
- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to
- **`timestamp`** (*`float`*) – Message timestamp

4.14 `platypush.message.response.tensorflow`

class `platypush.message.response.tensorflow.`**`TensorflowPredictResponse`**(**args, prediction: <sphinx.ext.autodoc.importer._ object at 0x7fc391748210>, output_labels: Optional[List[str]] = None, **kwargs*)

Tensorflow model prediction response.

`__init__`(**args, prediction: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc391748210>, output_labels: Optional[List[str]] = None, **kwargs*)

Parameters **`model`** – Name of the model.

class `platypush.message.response.tensorflow.`**`TensorflowResponse`**(**args, model: <sphinx.ext.autodoc.importer._MockObj object at 0x7fc39173c350>, model_name: Optional[str] = None, **kwargs*)

Generic Tensorflow response.

`__init__`(**args, model: <sphinx.ext.autodoc.importer._MockObject object at 0x7fc39173c350>, model_name: Optional[str] = None, **kwargs*)

Parameters **`model`** – Name of the model.

class `platypush.message.response.tensorflow.`**`TensorflowTrainResponse`**(**args, epochs: List[int], history: Dict[str, List[Union[int, float]]], **kwargs*)

Tensorflow model fit/train response.

`__init__`(**args, epochs: List[int], history: Dict[str, List[Union[int, float]]], **kwargs*)

Parameters

- **`epochs`** – List of epoch indexes the model has been trained on.
- **`history`** – Train history, as a `metric -> [values]` dictionary where each value in `values` is the value for of that metric on a specific epoch.

4.15 `platypush.message.response.todoist`

class `platypush.message.response.todoist.`**`TodoistCollaborator`**(*data: Dict[str, Any], *args, **kwargs*)

`__init__`(*data: Dict[str, Any], *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class `platypush.message.response.todoist.`**`TodoistCollaboratorsResponse`**(*collaborators: List[platypush.message.respon **kwargs*)

`__init__`(*collaborators: List[platypush.message.response.todoist.TodoistCollaborator], **kwargs*)

Parameters

- **`target`** (*`str`*) – Target
- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to

- **timestamp** (*float*) – Message timestamp

class platypush.message.response.todoist.**TodoistFilter**(*color: [<class 'int'>], id: [<class 'int'>], is_deleted: [<class 'bool'>], is_favorite: [<class 'bool'>], item_order: [<class 'int'>], name: [<class 'str'>], query: [<class 'str'>], legacy_id: Optional[int] = None, *args, **kwargs*)

__init__(*color: [<class 'int'>], id: [<class 'int'>], is_deleted: [<class 'bool'>], is_favorite: [<class 'bool'>], item_order: [<class 'int'>], name: [<class 'str'>], query: [<class 'str'>], legacy_id: Optional[int] = None, *args, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.todoist.**TodoistFiltersResponse**(*filters: List[platypush.message.response.todois **kwargs*)

__init__(*filters: List[platypush.message.response.todoist.TodoistFilter], **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.todoist.**TodoistItem**(*content: str, id: int, checked: bool, priority: int, child_order: int, collapsed: bool, day_order: int, date_added: datetime.datetime, in_history: bool, is_deleted: bool, user_id: int, has_more_notes: bool = False, project_id: Optional[int] = None, parent_id: Optional[int] = None, responsible_uid: Optional[int] = None, date_completed: Optional[datetime.datetime] = None, assigned_by_uid: Optional[int] = None, due: Optional[Dict[str, Any]] = None, labels: Optional[List[str]] = None, legacy_project_id: Optional[int] = None, section_id: Optional[int] = None, sync_id: Optional[int] = None, *args, **kwargs*)

`__init__`(*content: str*, *id: int*, *checked: bool*, *priority: int*, *child_order: int*, *collapsed: bool*, *day_order: int*, *date_added: datetime.datetime*, *in_history: bool*, *is_deleted: bool*, *user_id: int*, *has_more_notes: bool = False*, *project_id: Optional[int] = None*, *parent_id: Optional[int] = None*, *responsible_uid: Optional[int] = None*, *date_completed: Optional[datetime.datetime] = None*, *assigned_by_uid: Optional[int] = None*, *due: Optional[Dict[str, Any]] = None*, *labels: Optional[List[str]] = None*, *legacy_project_id: Optional[int] = None*, *section_id: Optional[int] = None*, *sync_id: Optional[int] = None*, **args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class `platypush.message.response.todoist.`**`TodoistItemsResponse`**(*items: List[platypush.message.response.todoist.To* ***kwargs*)

`__init__`(*items: List[platypush.message.response.todoist.TodoistItem]*, ***kwargs*)

Parameters

- **`target`** (*`str`*) – Target
- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to
- **`timestamp`** (*`float`*) – Message timestamp

class platypush.message.response.todoist.**TodoistLiveNotification**(*id: [<class 'int'>], is_deleted: [<class 'bool'>], created: [<class 'str'>], is_unread: [<class 'bool'>], notification_key: [<class 'str'>], notification_type: [<class 'str'>], completed_last_month: Optional[int] = None, karma_level: Optional[int] = None, promo_img: Optional[str] = None, completed_tasks: Optional[int] = None, *args, **kwargs*)

__init__(*id: [<class 'int'>], is_deleted: [<class 'bool'>], created: [<class 'str'>], is_unread: [<class 'bool'>], notification_key: [<class 'str'>], notification_type: [<class 'str'>], completed_last_month: Optional[int] = None, karma_level: Optional[int] = None, promo_img: Optional[str] = None, completed_tasks: Optional[int] = None, *args, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.todoist.**TodoistLiveNotificationsResponse**(*notifications: List[platypush.message.r **kwargs*)

__init__(*notifications: List[platypush.message.response.todoist.TodoistLiveNotification], **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.todoist.**TodoistNote**(*data: Dict[str, Any], *args, **kwargs*)

__init__(*data: Dict[str, Any], *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.todoist.**TodoistNotesResponse**(*notes: List[platypush.message.response.todoist.To **kwargs*)

__init__(*notes: List[platypush.message.response.todoist.TodoistCollaborator], **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.todoist.**TodoistProject**(*child_order: int, collapsed: int, color: int, has_more_notes: bool, id: int, is_archived: bool, is_deleted: bool, is_favorite: bool, name: str, shared: bool, inbox_project: Optional[bool] = None, legacy_id: Optional[int] = None, parent_id: Optional[int] = None, *args, **kwargs*)

__init__(*child_order: int, collapsed: int, color: int, has_more_notes: bool, id: int, is_archived: bool, is_deleted: bool, is_favorite: bool, name: str, shared: bool, inbox_project: Optional[bool] = None, legacy_id: Optional[int] = None, parent_id: Optional[int] = None, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.todoist.**TodoistProjectNote**(*data: Dict[str, Any], *args, **kwargs*)

__init__(*data: Dict[str, Any], *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.todoist.**TodoistProjectNotesResponse**(*notes: List[platypush.message.response **kwargs*)

__init__(*notes: List[platypush.message.response.todoist.TodoistCollaborator], **kwargs*)

Parameters

- **target** (*str*) – Target

- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to
- **`timestamp`** (*`float`*) – Message timestamp

class `platypush.message.response.todoist.`**`TodoistProjectsResponse`**(*projects: List[platypush.message.response.todo **kwargs*)

`__init__`(*projects: List[platypush.message.response.todoist.TodoistProject], **kwargs*)

Parameters

- **`target`** (*`str`*) – Target
- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to
- **`timestamp`** (*`float`*) – Message timestamp

class `platypush.message.response.todoist.`**`TodoistReminder`**(*data: Dict[str, Any], *args, **kwargs*)

`__init__`(*data: Dict[str, Any], *args, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class `platypush.message.response.todoist.`**`TodoistRemindersResponse`**(*reminders: List[platypush.message.response.tod **kwargs*)

`__init__`(*reminders: List[platypush.message.response.todoist.TodoistReminder], **kwargs*)

Parameters

- **`target`** (*`str`*) – Target
- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to
- **`timestamp`** (*`float`*) – Message timestamp

class `platypush.message.response.todoist.`**`TodoistResponse`**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

class platypush.message.response.todoist.**TodoistUserResponse**(*auto_reminder: Optional[int] = None, avatar_big: Optional[str] = None, avatar_medium: Optional[str] = None, avatar_s640: Optional[str] = None, avatar_small: Optional[str] = None, business_account_id: Optional[int] = None, daily_goal: Optional[int] = None, date_format: Optional[str] = None, dateist_inline_disabled: Optional[bool] = None, dateist_lang: Optional[str] = None, days_off: Optional[List[int]] = None, default_reminder: Optional[str] = None, email: Optional[str] = None, features: Optional[Dict[str, Any]] = None, full_name: Optional[str] = None, id: Optional[int] = None, image_id: Optional[str] = None, inbox_project: Optional[int] = None, is_biz_admin: Optional[bool] = None, is_premium: Optional[bool] = None, join_date: Optional[datetime.datetime] = None, karma: Optional[float] = None, karma_trend: Optional[str] = None, lang: Optional[str] = None, legacy_inbox_project: Optional[int] = None, mobile_host: Optional[str]*

__init__(*auto_reminder: Optional[int] = None*, *avatar_big: Optional[str] = None*, *avatar_medium: Optional[str] = None*, *avatar_s640: Optional[str] = None*, *avatar_small: Optional[str] = None*, *business_account_id: Optional[int] = None*, *daily_goal: Optional[int] = None*, *date_format: Optional[str] = None*, *dateist_inline_disabled: Optional[bool] = None*, *dateist_lang: Optional[str] = None*, *days_off: Optional[List[int]] = None*, *default_reminder: Optional[str] = None*, *email: Optional[str] = None*, *features: Optional[Dict[str, Any]] = None*, *full_name: Optional[str] = None*, *id: Optional[int] = None*, *image_id: Optional[str] = None*, *inbox_project: Optional[int] = None*, *is_biz_admin: Optional[bool] = None*, *is_premium: Optional[bool] = None*, *join_date: Optional[datetime.datetime] = None*, *karma: Optional[float] = None*, *karma_trend: Optional[str] = None*, *lang: Optional[str] = None*, *legacy_inbox_project: Optional[int] = None*, *mobile_host: Optional[str] = None*, *mobile_number: Optional[str] = None*, *next_week: Optional[int] = None*, *premium_until: Optional[datetime.datetime] = None*, *share_limit: Optional[int] = None*, *sort_order: Optional[int] = None*, *start_day: Optional[int] = None*, *start_page: Optional[str] = None*, *theme: Optional[int] = None*, *time_format: Optional[int] = None*, *token: Optional[str] = None*, *tz_info: Optional[Dict[str, Any]] = None*, *unique_prefix: Optional[int] = None*, *websocket_url: Optional[str] = None*, *weekly_goal: Optional[int] = None*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.16 platypush.message.response.translate

class platypush.message.response.translate.**TranslateResponse**(*translated_text: str*, *source_text: str*, *detected_source_language: str*, **args*, ***kwargs*)

__init__(*translated_text: str*, *source_text: str*, *detected_source_language: str*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

4.17 `platypush.message.response.trello`

class platypush.message.response.trello.**TrelloAttachment**(*id: str, bytes: int, date: str, edge_color: str, id_member: str, is_upload: bool, name: str, previews: List[platypush.message.response.trello.TrelloPrev url: str, mime_type: Optional[str] = None, *args, **kwargs*)

__init__(*id: str, bytes: int, date: str, edge_color: str, id_member: str, is_upload: bool, name: str, previews: List[platypush.message.response.trello.TrelloPreview], url: str, mime_type: Optional[str] = None, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloBoard**(*id: str, name: str, url: str, closed: bool, lists: Optional[List[platypush.message.response.trello.TrelloList]] = None, description: Optional[str] = None, date_last_activity: Optional[datetime.datetime] = None, *args, **kwargs*)

__init__(*id: str, name: str, url: str, closed: bool, lists: Optional[List[platypush.message.response.trello.TrelloList]] = None, description: Optional[str] = None, date_last_activity: Optional[datetime.datetime] = None, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloBoardResponse**(*board: platypush.message.response.trello.TrelloBoard, **kwargs*)

__init__(*board: platypush.message.response.trello.TrelloBoard, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.trello.**TrelloBoardsResponse**(*boards: List[platypush.message.response.trello.Trell **kwargs*)

__init__(*boards: List[platypush.message.response.trello.TrelloBoard], **kwargs*)

Parameters

- **target** (*str*) – Target

- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.trello.**TrelloCard**(*id: str, name: str, url: str, closed: bool, board: platypush.message.response.trello.TrelloBoard, is_due_complete: bool, list: Optional[platypush.message.response.trello.TrelloList] = None, comments: Optional[List[platypush.message.response.trello.TrelloComme = None, labels: Optional[List[platypush.message.response.trello.TrelloLabel] = None, description: Optional[str] = None, due_date: Union[datetime.datetime, str, None] = None, latest_card_move_date: Union[datetime.datetime, str, None] = None, date_last_activity: Union[datetime.datetime, str, None] = None, *args, **kwargs*)

__init__(*id: str, name: str, url: str, closed: bool, board: platypush.message.response.trello.TrelloBoard, is_due_complete: bool, list: Optional[platypush.message.response.trello.TrelloList] = None, comments: Optional[List[platypush.message.response.trello.TrelloComment]] = None, labels: Optional[List[platypush.message.response.trello.TrelloLabel]] = None, description: Optional[str] = None, due_date: Union[datetime.datetime, str, None] = None, latest_card_move_date: Union[datetime.datetime, str, None] = None, date_last_activity: Union[datetime.datetime, str, None] = None, *args, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloCardResponse**(*card: platypush.message.response.trello.TrelloCard, **kwargs*)

__init__(*card: platypush.message.response.trello.TrelloCard, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.trello.**TrelloCardsResponse**(*cards: List[platypush.message.response.trello.Trello* ***kwargs*)

__init__(*cards: List[platypush.message.response.trello.TrelloCard], **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.trello.**TrelloChecklist**(*id: str, name: str, checklist_items: List[platypush.message.response.trello.TrelloCheck* **args, **kwargs*)

__init__(*id: str, name: str, checklist_items: List[platypush.message.response.trello.TrelloChecklistItem], *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloChecklistItem**(*id: str, name: str, checked: bool, *args, **kwargs*)

__init__(*id: str, name: str, checked: bool, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloComment**(*id: str, text: str, type: str, creator: platypush.message.response.trello.TrelloUser, date: Union[str, datetime.datetime], *args, **kwargs*)

__init__(*id: str, text: str, type: str, creator: platypush.message.response.trello.TrelloUser, date: Union[str, datetime.datetime], *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloLabel**(*id: str, name: str, color: Optional[str] = None, *args, **kwargs*)

__init__(*id: str, name: str, color: Optional[str] = None, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloList**(*id: str, name: str, closed: bool, subscribed: bool, *args, **kwargs*)

__init__(*id: str, name: str, closed: bool, subscribed: bool, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloListsResponse**(*lists: List[platypush.message.response.trello.Trello* ***kwargs*)

__init__(*lists: List[platypush.message.response.trello.TrelloList], **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.trello.**TrelloMember**(*id: str, full_name: str, bio: Optional[str], url: Optional[str], username: Optional[str], initials: Optional[str], member_type: Optional[str] = None, *args, **kwargs*)

__init__(*id: str, full_name: str, bio: Optional[str], url: Optional[str], username: Optional[str], initials: Optional[str], member_type: Optional[str] = None, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloMembersResponse**(*members: List[platypush.message.response.trello.Tre* ***kwargs*)

__init__(*members: List[platypush.message.response.trello.TrelloMember], **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloPreview**(*id: str, scaled: bool, url: str, bytes: int, height: int, width: int, *args, **kwargs*)

__init__(*id: str, scaled: bool, url: str, bytes: int, height: int, width: int, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.trello.**TrelloResponse**(*target=None, origin=None, id=None, output=None, errors=None, timestamp=None, disable_logging=False*)

class platypush.message.response.trello.**TrelloUser**(*id: str, username: str, fullname: str, initials: Optional[str] = None, avatar_url: Optional[str] = None, *args, **kwargs*)

__init__(*id: str, username: str, fullname: str, initials: Optional[str] = None, avatar_url: Optional[str] = None, *args, **kwargs*)
Initialize self. See help(type(self)) for accurate signature.

4.18 `platypush.message.response.weather.buienradar`

class platypush.message.response.weather.buienradar.**BuienradarForecast**(*condition_name: str, condition_name_long: str, condition_image: str, date_time: datetime.datetime, rain: float, min_rain: float, max_rain: float, rain_chance: float, snow: int, temperature: float, wind_azimuth: int, wind_direction: str, wind_force: int, wind_speed: float, *args, **kwargs*)

__init__(*condition_name: str, condition_name_long: str, condition_image: str, date_time: datetime.datetime, rain: float, min_rain: float, max_rain: float, rain_chance: float, snow: int, temperature: float, wind_azimuth: int, wind_direction: str, wind_force: int, wind_speed: float, *args, **kwargs*)

Initialize self. See help(type(self)) for accurate signature.

class platypush.message.response.weather.buienradar.**BuienradarForecastResponse**(*forecast=typing.I*
**args, **kwargs*)

__init__(*forecast=typing.List[platypush.message.response.weather.buienradar.BuienradarForecast], *args, **kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.weather.buienradar.**BuienradarPrecipitationResponse**(*average: float*, *to-tal: float*, *time_fram int*, **args*, ***kwargs*

__init__(*average: float*, *total: float*, *time_frame: int*, **args*, ***kwargs*)

Parameters

- **target** (*str*) – Target
- **origin** (*str*) – Origin
- **output** – Output
- **errors** – Errors
- **id** (*str*) – Message ID this response refers to
- **timestamp** (*float*) – Message timestamp

class platypush.message.response.weather.buienradar.**BuienradarWeatherResponse**(*barometer_fc: str*, *condition_name: str*, *condition_name_long: str*, *condition_image: str*, *feel_temperature: float*, *ground_temperatu float*, *humidity: int*, *irradiance: int*, *measured: datetime.datetime*, *precipitation: float*, *pressure: float*, *rain_last_24_hour float*, *rain_last_hour: float*, *station_name: str*, *temperature: float*, *visibility: int*, *wind_azimuth: int*, *wind_direction:*

`__init__`(*barometer_fc: str*, *condition_name: str*, *condition_name_long: str*, *condition_image: str*, *feel_temperature: float*, *ground_temperature: float*, *humidity: int*, *irradiance: int*, *measured: datetime.datetime*, *precipitation: float*, *pressure: float*, *rain_last_24_hours: float*, *rain_last_hour: float*, *station_name: str*, *temperature: float*, *visibility: int*, *wind_azimuth: int*, *wind_direction: str*, *wind_force: int*, *wind_gust: float*, *wind_speed: float*, **args*, ***kwargs*)

Parameters

- **`target`** (*`str`*) – Target
- **`origin`** (*`str`*) – Origin
- **`output`** – Output
- **`errors`** – Errors
- **`id`** (*`str`*) – Message ID this response refers to
- **`timestamp`** (*`float`*) – Message timestamp

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index