PyWiliot-api

PyWiliot-api is a Python library allowing users to use Wiliot's APIs. This guide provides PyWiliot usage information.

Content

Installation

To install PyWiliot run

```
pip install wiliot-api
```

Usage

Authentication

PyWiliot handles the authentication and authorization required by Wiliot APIs. The user must provide PyWiliot with their Wiliot account credentials

Owner ID (Account)

Wiliot IoT pixels and other online assets are managed under owners (accounts). PyWiliot must be given an owner ID when instantiated. See examples below

Platform API

Create a client for the Platform API - you'll need to provide an API key. One can be generated following the instructions in this guide

```
from wiliot_api.platform.platform import *

my_api_key = "your API key copied from the platform"
owner = "my_company"

platform = PlatformClient(api_key=my_api_key, owner_id=owner)
```

Following are examples of common usages of Wiliot's Platform API

Pixels

Get a list of all an owner's (account's) pixels.

```
platform.get_pixels()
```

Called like that the API will return a tuple containing two values:

- A list of the first 50 pixels
- The ID of the next pixel (the 51st one)

To get a list of all pixel IDs under an account (not recommended for accounts with a large number of pixels), use this code

```
pixels = []
next_p = None
while True:
    p, next_p = platform.get_pixels(next=next_p)
    pixels += [p1['id'] for p1 in p]
    if next_p is None:
        break
```

Categories

See more information about categories here

Get a list of all categories

```
platform.get_categories()
```

Each category is based on an asset type, which dictates the minimum number of pixels that need to be applied to assets belonging to the category and the events generated for those assets. To get a list of all asset types, use this code

```
platform.get_asset_types()
```

To create a category, provide:

- A base asset type ID
- A category name
- A category ID (optional - if none is provided an ID will be created automatically. This can be used to record SKU, for example)
- Description (optional - visible in the platform Web UI)

The following code will create a new category

```
platform.create_category(name="Brown_folding_crate",
                         asset_type_id="Default",
                         category_id="brown-folding-crate",
                         description="A folding brown crate")
```

Assets

Assets represent any item with one or more pixels attached. Read more about assets here .

Get a list of all assets - warning, this call could take a long time to complete on accounts with a large number of assets.

```
platform.get_assets()
```

Get one asset using its ID.

```
platform.get_asset("some-asset-id")
```

Create an asset and assign one or more pixels to it. For each pixel, provide an ID and a role (for tracked assets, the role is DEFAULT)

```
platform.create_asset(name="brown_folding_crate_116872",
                      category_id="brown-folding-crate",
                      asset_id="brown-folding-crate-116872",
                      pixels=[{"tagId": "pixel-1-id", "role": TagRole.DEFAULT},
                              {"tagId": "pixel-2-id", "role": TagRole.DEFAULT}])
```

Associate a pixel to an existing asset

```
platform.associate_pixel_to_asset(asset_id)
```

Asset Labels

Assets can have one or more labels associated with them. Each label is limited to 12 alphanumeric characters (no symbols). The following code adds a label to an asset

```
platform.create_asset_label(asset_id="my-asset-id", label="myassetlabel")
```

Note: All asset label function calls include a "project_id" parameter, which defaults to "Default". To ensure compatibility with future API versions, it is recommended to use this default value, so there is no need to pass it to any asset label function.

To get all asset labels, use the following code:

```
platform.get_asset_labels(asset_id="my-asset-id")
```

The following code will delete one asset label

```
platform.delete_asset_label(asset_id="my-asset-id", label="myassetlabel")
```

To delete all asset labels, use the following code

```
platform.delete_asset_labels(asset_id="my-asset-id")
```

Locations and Zones

For more information about locations and zones, see this guide .

To get a list of all locations defined under the account, use the following example

```
platform.get_locations()
```

Details about a single location can be retrieved using the following example

```
platform.get_location(location_id)
```

To create a new location, provide:

- Location type - one of:
  - LocationType.SITE - for fixed locations (for example, warehouse)
  - LocationType.TRANSPORTER - for mobile locations (for example, truck)

- Location name - a humanly readable name for the location (optional)
- Latitude - optional, for fixed locations
- Longitude - optional, for fixed locations
- Address - optional, for fixed locations
- City - optional, for fixed locations
- Country - optional, for fixed locations

Note: It's possible to provide a location ID as well. If one is not provided, the location will be given an automatically generated ID. The following code will create a new location

```
platform.create_location(location_type=LocationType.SITE,
                         location_name="My new location",
                         lat=32.4854738,
                         lng=30.46452)
```

Zones are used to designate a smaller area within a location, for example, a room in a warehouse. Therefore, every operation on zones requires a location ID. To get a list of all zones under a location, use the following code:

```
platform.get_zones(location_id="some-location-id")
```

To create a new zone provide:

- A location ID - An ID the zone will belong to
- Name - A humanly readable name for the zone

Note: It's possible to provide a zone ID as well. If one is not provided, the zone will be given an automatically generated ID. The following code creates a new zone under an existing location:

```
platform.create_zone(name="Room 1",
                     location_id="warehouse-1-location-id")
```

Bridges can be associated with zones or locations. Once a bridge is associated with a zone, location events generated by the Wiliot cloud will include the associated zone ID. To list all of the associations for a zone, use the following code:

```
platform.get_zone_associations(zone_id="room-1-zone-id")
```

To create a new zone association, provide:

- Location ID
- Zone ID
- Association Type - Currently can only be ZoneAssociationType.BRIDGE
- Association Value - The Bridge ID

```
platform.create_zone_association(location_id="warehouse-1-location-id",
                                 zone_id="room-1-zone-id",
                                 association_type=ZoneAssociationType.BRIDGE,
                                 association_value"bridge-id")
```

Delete a single zone association using the following code:

```
platform.delete_zone_association(location_id="warehouse-1-location-id",
                                 zone_id="room-1-zone-id",
                                 association_value="bridge-id")
```

Delete all zone associations for a single zone (this is required before deleting a zone):

```
platform.delete_zone_associations(location_id="warehouse-1-location-id",
                                  zone_id="room-1-zone-id")
```

Edge API

The Edge API allows access to all edge-device (bridges and gateways) related operations. Create a client using the following code

```
from wiliot_api.edge.edge import *

my_api_key = "your Edge API key copied from the platform"
owner = "my_company"

edge = EdgeClient(api_key=my_api_key, owner_id=owner)
```

Gateways

Get a list of owner's (account's) gateways

```
edge.get_gateways()
```

Get detailed information about one gateway

```
edge.get_gateway("gateway_id")
```

Register one or more Wiliot gateways

```
edge.register_gateway(["gateway_id"])
```

Approve a Wiliot gateway

```
edge.approve_gateway("gateway_id")
```

Update a gateway's configuration - requires providing a configuration dictionary as well as a list of gateway IDs to update. Refer to this guide for details about available configuration options. The following code demonstrates a common update sequence

```
existing_conf = w.get_gateway_details("gateway_id")
desired_conf = existing_conf["desiredConf"]

# Update one setting - in this case, the pacer interval
desired_conf["additional"]["pacerInterval"] = 60

edge.update_gateway_configuration(["gateway_id"], desired_conf)
```

Bridges

Get a list of bridges owned (claimed) by the account

```
edge.get_bridges()
```

Get information about a specific bridge

```
edge.get_bridge("bridge_id")
```

Claim a bridge

```
edge.claim_bridge("bridge_id")
```

Unclaim a bridge

```
edge.unclaim_bridge("bridge_id")
```

Request an update of a bridge's configuration

```
# Get the bridge's current configuration
config = wiliot.get_bridge("bridge_id")["config"]
# Make the desired configuration change
config["pacerInterval"] = 60
# Request configuration update
wiliot.update_bridge_configuration("bridge_id", config)
```

Send an action to a bridge. Two actions are supported:

- Reboot
- Blink LED

Note: The bridge must be online for the call to succeed

```
from wiliot_api.edge.edge import BridgeAction

# Make a bridge blink its LED
edge.send_action_to_bridge(bridge_id="my-bridge-id", action=BridgeAction.BLINK_LED)

# Reboot a bridge
edge.send_action_to_bridge(bridge_id="my-bridge-id", action=BridgeAction.REBOOT)
```