adafruit-io-python Library Documentation

Release 2.1.0

Adafruit Industries

Apr 20, 2022

Contents

A Python library and examples for use with io.adafruit.com.

Compatible with Python Versions 3.4+

CHAPTER 1

Installation

1.1 Easy Installation

If you have PIP installed (typically with apt-get install python-pip on a Debian/Ubuntu-based system) then run:

```
pip3 install adafruit-io
```

This will automatically install the Adafruit IO Python client code for your Python scripts to use. You might want to examine the examples folder in this GitHub repository to see examples of usage.

If the above command fails, you may first need to install prerequisites:

```
pip3 install setuptools
pip3 install wheel
```

1.2 Manual Installation

Clone or download the contents of this repository. Then navigate to the folder in a terminal and run the following command:

```
python setup.py install
```

CHAPTER 2

Usage

Documentation for this project is available on the ReadTheDocs.

CHAPTER 3

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 4

Table of Contents

4.1 Quickstart

Here's a short example of how to send a new value to a feed (creating the feed if it doesn't exist), and how to read the most recent value from the feed. This example uses the REST API.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT USER', 'YOUR ADAFRUIT IO KEY')

# Send the value 100 to a feed called 'Foo'.
aio.send('Foo', 100)

# Retrieve the most recent value from the feed 'Foo'.
# Access the value by reading the `value` property on the returned Data object.
# Note that all values retrieved from IO are strings so you might need to convert
# them to an int or numeric type if you expect a number.
data = aio.receive('Foo')
print('Received value: {0}'.format(data.value))
```

If you want to be notified of feed changes immediately without polling, consider using the MQTT client. See the `examples/mqtt_client.py` for an example of using the MQTT client.

4.2 Basic Client Usage

You must have an Adafruit IO key to use this library and the Adafruit IO service. Your API key will be provided to the python library so it can authenticate your requests against the Adafruit IO service.

At a high level the Adafruit IO python client provides two interfaces to the service:

- A thin wrapper around the REST-based API. This is good for simple request and response applications like logging data.

- A MQTT client (based on paho-mqtt) which can publish and subscribe to feeds so it is immediately alerted of changes. This is good for applications which need to know when something has changed as quickly as possible.

To use either interface you'll first need to import the python client by adding an import such as the following at the top of your program:

```
from Adafruit_IO import *
```

Then a REST API client can be created with code like:

```
aio = Client('user', 'xxxxxxxxxxxx')
```

Where `'xxxxxxxxxxxx'` is your Adafruit IO API key. Where `'user'` is your Adafruit username.

Alternatively an MQTT client can be created with code like:

```
mqtt = MQTTClient('user', 'xxxxxxxxxxxx')
```

Again where `'xxxxxxxxxxxx'` is your Adafruit IO API key. Again where `'user'` is your Adafruit username.

Your program can use either or both the REST API client and MQTT client, depending on your needs.

4.3 Error Handling

The python client library will raise an exception if it runs into an error it cannot handle. You should be prepared to catch explicit exceptions you know how to handle, or bubble them up to the user as an error. Adafruit IO exceptions generally are children of the base exception type AdafruitIOError. There are also three sub-exceptions to handle, depending on which if you're using the REST API or MQTT Client: MQTTError (for the MQTT Client), RequestError (REST Client), and ThrottlingError (REST Client).

4.4 Feeds

Feeds are the core of the Adafruit IO system. The feed holds metadata about data that gets pushed, and you will have one feed for each type of data you send to the system. You can have separate feeds for each sensor in a project, or you can use one feed to contain JSON encoded data for all of your sensors.

4.4.1 Feed Creation

Create a feed by constructing a Feed instance with at least a name specified, and then pass it to the `create_feed(feed)` function:

```
# Import library and create instance of REST client.
from Adafruit_IO import Client, Feed
aio = Client('YOUR ADAFRUIT IO KEY')

# Create Feed object with name 'Foo'.
feed = Feed(name='Foo')

# Send the Feed to IO to create.
# The returned object will contain all the details about the created feed.
result = aio.create_feed(feed)
```

Note that you can use the send function to create a feed and send it a new value in a single call. It's recommended that you use send instead of manually constructing feed instances.

4.4.2 Feed Retrieval

You can get a list of your feeds by using the `feeds()` method which will return a list of Feed instances:

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO KEY')

# Get list of feeds.
feeds = aio.feeds()

# Print out the feed names:
for f in feeds:
    print('Feed: {0}'.format(f.name))
```

Alternatively you can retrieve the metadata for a single feed by calling `feeds(feed)` and passing the `name`, `ID`, or `key` of a feed to retrieve:

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO KEY')

# Get feed 'Foo'
feed = aio.feeds('Foo')

# Print out the feed metadata.
print(feed)
```

4.4.3 Feed Deletion

You can delete a feed by ID, key, or name by calling `delete_feed(feed)`. ALL data in the feed will be deleted after calling this API!

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Delete the feed with name 'Test'.
aio.delete_feed('Test')
```

4.5 Feed Sharing

Feed sharing is a feature of Adafruit IO which allows you to share your feeds with people you specify.

If you want to share a feed on your Adafruit IO Account with another user, visit the Sharing a feed page on the Adafruit Learning System.

The Adafruit IO Python client supports Feed Sharing in the mqtt_client.py class.

4.5.1 Usage Example

```
"""
`mqtt_shared_feeds.py`
--------------------------------------------------------
Example of reading and writing to a shared  Adafruit IO Feed.

learn.adafruit.com/adafruit-io-basics-feeds/sharing-a-feed

Author: Brent Rubell for Adafruit Industries 2018
"""

# Import standard python modules.
import sys
import time
import random

# Import Adafruit IO MQTT client.
from Adafruit_IO import MQTTClient

# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_AIO_KEY'

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_AIO_USERNAME'

# Shared IO Feed
# Make sure you have read AND write access to this feed to publish.
IO_FEED = 'SHARED_AIO_FEED_NAME'

# IO Feed Owner's username
IO_FEED_USERNAME = 'SHARED_AIO_FEED_USERNAME'


# Define callback functions which will be called when certain events happen.
def connected(client):
    """Connected function will be called when the client connects.
    """
    client.subscribe(IO_FEED, IO_FEED_USERNAME)

def disconnected(client):
    """Disconnected function will be called when the client disconnects.
    """
    print('Disconnected from Adafruit IO!')
    sys.exit(1)

def message(client, feed_id, payload):
    """Message function will be called when a subscribed feed has a new value.
    The feed_id parameter identifies the feed, and the payload parameter has
    the new value.
    """
    print('Feed {0} received new value: {1}'.format(feed_id, payload))


# Create an MQTT client instance.
client = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)
```

(continues on next page)

(continued from previous page)

```
# Setup the callback functions defined above.
client.on_connect       =   connected
client.on_disconnect    =   disconnected
client.on_message       =   message

# Connect to the Adafruit IO server.
client.connect()

client.loop_background()
print('Publishing a new message every 10 seconds (press Ctrl-C to quit)...')

while True:
    value = random.randint(0, 100)
    print('Publishing {0} to {1}.'.format(value, IO_FEED))
    client.publish(IO_FEED, value, IO_FEED_USERNAME)
    time.sleep(10)
```

4.6 Data

Data represents the data contained in feeds. You can read, add, modify, and delete data. There are also a few convenient methods for sending data to feeds and selecting certain pieces of data.

4.6.1 Data Creation

Data can be created after you create a feed, by using the `create_data(feed, data)` method and passing it a new Data instance a value.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client, Data
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Create a data item with value 10 in the 'Test' feed.
data = Data(value=10)
aio.create_data('Test', data)
```

4.6.2 Data Retrieval

You can get all of the data for a feed by using the `data(feed)` method. The result will be an array of all feed data, each returned as an instance of the Data class. Use the value property on each Data instance to get the data value, and remember values are always returned as strings (so you might need to convert to an int or number if you expect a numeric value).

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Get an array of all data from feed 'Test'
data = aio.data('Test')

# Print out all the results.
```

(continues on next page)

(continued from previous page)

```
for d in data:
    print('Data value: {0}'.format(d.value))
```

By default, the maximum number of data points returned is 1000. This limit can be changed by using the max_results parameter.

```
# Get less than the default number of data points
data = aio.data('Test', max_results=100)

# Get more than the default number of data points
data = aio.data('Test', max_results=2000)

# Get all of the points
data = aio.data('Test', max_results=None)
```

You can also get a specific value by ID by using the `feeds(feed, data_id)` method. This will return a single piece of feed data with the provided data ID if it exists in the feed. The returned object will be an instance of the Data class.

4.6.3 Data Deletion

Values can be deleted by using the `delete(feed, data_id)` method:

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Delete a data value from feed 'Test' with ID 1.
data = aio.delete('Test', 1)
```

4.7 Data Helper methods

There are a few helper methods that can make interacting with data a bit easier.

4.7.1 Send Data

You can use the `send_data(feed_name, value)` method to append a new value to a feed. This is the recommended way to send data to Adafruit IO from the Python REST client.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Add the value 98.6 to the feed 'Temperature'.
test = aio.feeds('test')
aio.send_data(test.key, 98.6)
```

4.7.2 Send Batch Data

Data can be created after you create a feed, by using the `send_batch_data(feed, data_list)` method and passing it a new Data list.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client, Data
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Create a data items in the 'Test' feed.
data_list = [Data(value=10), Data(value=11)]
aio.create_data('Test', data)
```

4.7.3 Receive Data

You can get the last inserted value by using the `receive(feed)` method.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Get the last value of the temperature feed.
data = aio.receive('Test')

# Print the value and a message if it's over 100.  Notice that the value is
# converted from string to int because it always comes back as a string from IO.
temp = int(data.value)
print('Temperature: {0}'.format(temp))
if temp > 100:
    print 'Hot enough for you?'
```

4.7.4 Next Value

You can get the first inserted value that has not been processed (read) by using the `receive_next(feed)` method.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Get next unread value from feed 'Test'.
data = aio.receive_next('Test')

# Print the value.
print('Data value: {0}'.format(data.value))
```

4.7.5 Previous Value

You can get the last record that has been processed (read) by using the `receive_previous(feed)` method.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

```

(continues on next page)

(continued from previous page)

```
# Get previous read value from feed 'Test'.
data = aio.receive_previous('Test')

# Print the value.
print('Data value: {0}'.format(data.value))
```

4.7.6 Publishing and Subscribing

You can get a readable stream of live data from your feed using the included MQTT client class:

```
# Example of using the MQTT client class to subscribe to a feed and print out
# any changes made to the feed.  Edit the variables below to configure the key,
# username, and feed to subscribe to for changes.

# Import standard python modules.
import sys

# Import Adafruit IO MQTT client.
from Adafruit_IO import MQTTClient

# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_AIO_KEY'

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_AIO_USERNAME'

# Set to the ID of the feed to subscribe to for updates.
FEED_ID = 'DemoFeed'


# Define callback functions which will be called when certain events happen.
def connected(client):
    # Connected function will be called when the client is connected to Adafruit IO.
    # This is a good place to subscribe to feed changes.  The client parameter
    # passed to this function is the Adafruit IO MQTT client so you can make
    # calls against it easily.
    print('Connected to Adafruit IO!  Listening for {0} changes...'.format(FEED_ID))
    # Subscribe to changes on a feed named DemoFeed.
    client.subscribe(FEED_ID)

def subscribe(client, userdata, mid, granted_qos):
    # This method is called when the client subscribes to a new feed.
    print('Subscribed to {0} with QoS {1}'.format(FEED_ID, granted_qos[0]))

def disconnected(client):
    # Disconnected function will be called when the client disconnects.
    print('Disconnected from Adafruit IO!')
    sys.exit(1)

def message(client, feed_id, payload):
    # Message function will be called when a subscribed feed has a new value.
    # The feed_id parameter identifies the feed, and the payload parameter has
```

(continues on next page)

(continued from previous page)

```
    # the new value.
    print('Feed {0} received new value: {1}'.format(feed_id, payload))


# Create an MQTT client instance.
client = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

# Setup the callback functions defined above.
client.on_connect    = connected
client.on_disconnect = disconnected
client.on_message    = message
client.on_subscribe  = subscribe

# Connect to the Adafruit IO server.
client.connect()

# Start a message loop that blocks forever waiting for MQTT messages to be
# received.  Note there are other options for running the event loop like doing
# so in a background thread--see the mqtt_client.py example to learn more.
client.loop_blocking()
```

4.8 Groups

Groups allow you to update and retrieve multiple feeds with one request. You can add feeds to multiple groups.

4.8.1 Group Creation

The creation of groups is now supported in API-V2, rejoyce! The process of creating a group is similar to creating a feed. Create a group by constructing a Group instance with at least a name specified, and then pass it to the `create_group(group)` function:

```
# Import library and create instance of REST client.
from Adafruit_IO import Client, Group
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Create a group instance
group = Group(name="weatherstation")

# Send the group for IO to create:
# The returned object will contain all the details about the created group.
group = aio.create_group(group)
```

4.8.2 Group Retrieval

You can get a list of your groups by using the `groups()` method. This will return a list of Group instances. Each Group instance has metadata about the group, including a `feeds` property which is a tuple of all feeds in the group.

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')
```

(continues on next page)

(continued from previous page)

```
# Get list of groups.
groups = aio.groups()

# Print the group names and number of feeds in the group.
for g in groups:
    print('Group {0} has {1} feed(s).'.format(g.name, len(g.feeds)))
```

You can also get a specific group by ID, key, or name by using the `groups(group)` method:

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Get group called 'GroupTest'.
group = aio.groups('GroupTest')

# Print the group name and number of feeds in the group.
print('Group {0} has {1} feed(s).'.format(group.name, len(group.feeds)))
```

4.8.3 Group Updating

TODO: Test and example this

4.8.4 Group Deletion

You can delete a group by ID, key, or name by using the `delete_group(group)` method:

```
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('YOUR ADAFRUIT IO USERNAME', 'YOUR ADAFRUIT IO KEY')

# Delete group called 'GroupTest'.
aio.delete_group('GroupTest')
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search