io-client-python Library Documentation

Release 2.0

Tony DiCola, Justin Cooper, Adam Bachman, Todd Treece, Brent

Jul 02, 2018

Contents

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

Compatible with Python 3.6+

Contents

Documentation

Documentation for this project is available on the ReadTheDocs.

CHAPTER 2

Contributing

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

Table of Contents

3.1 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.

```python
# Import library and create instance of REST client.
from Adafruit_IO import Client
aio = Client('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.

3.2 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:

```python
from Adafruit_IO import *
```

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

```python
aio = Client('xxxxxxxxxxxx')
```

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

Alternatively an MQTT client can be created with code like:

```python
mqtt = MQTTClient('xxxxxxxxxxxx')
```

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

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

3.3 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.

3.4 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.

3.4.1 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:

```python
# 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.

3.4.2 Feed Retrieval

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

```python
# 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:

```python
# 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)
```

3.4.3 Feed Retrieval

TODO: This is not tested in the python client yet, but calling create_feed with a Feed instance should update the feed.

3.4.4 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!

```python
# 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')
```

3.5 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.

3.5.1 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.

```python
# 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)
```

3.5.2 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).

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

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.

3.5.3 Data Retrieval

TODO: This is not tested in the python client, but calling `create_data` with a Data instance should update it.

3.5.4 Data Deletion

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

```python
# 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)
```

3.6 Data Helper methods

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

3.6.1 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.

```python
# 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)
```

3.6.2 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.

```python
# 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)
```

3.6.3 Receive Data

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

3.6.4 Next Value

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

```python
# 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))
```

3.6.5 Previous Value

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

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

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

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

3.6.6 Publishing and Subscribing

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

3.7 Groups

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

3.7.1 Group Creation

Groups must be created in the UI, group creation is unsupported with the API.

3.7.2 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.

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

# 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:

3.7.3 Group Updating

TODO: Test and example this

3.7.4 Group Deletion

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

```python
# 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')
```

Indices and tables

- genindex
- modindex
- search