tibber.py

Release 0.3.0

BeatsuDev

Sep 22, 2023

WELCOME

CHAPTER

ONE

INSTALLATION

tibber.py is available on PyPI. You can install it with pip:

```
pip install tibber.py
```

CHAPTER

TWO

GETTING BASIC INFORMATION

Getting the "Viewer" type (this is the topmost schema (or type) under Query on the Tibber API explorer documentation). All your homes, subscriptions, etc. will be under here.

```
import tibber

account = tibber.Account("your token")  # This is just like the viewer type in the API␣
↪explorer
```

Log in here: https://developer.tibber.com/explorer and check out the documentation on the right. Click the yellow underlined text "Query", then "Viewer". The "account" variable above is essentially the same as the Viewer type in the API explorer. It contains the same properties as the documentation says that the Viewer has. So let's for example get the name of the Viewer!

```
print(account.name)
```

What about getting a home? Checking out the https://developer.tibber.com/explorer documentation, we see that the Viewer type has a "homes" attribute and is surrounded by [brackets]. That means it comes in the form of a list. Let's get the first home in the list!

```
print(account.homes)  # [<Home: Home 1>, <Home: Home 2>, ...]
home = account.homes[0]
print(home)  # <Home: Home 1>
```

CHAPTER

THREE

RETRIEVING CONSUMPTION/PRODUCTION INFORMATION

Moving on from the previous example, what can we do with a home? Let's check out the documentation again. Click the yellow underlined text "Home" and check out the documentation. We see that the Home type has a "consumption" method! Let's try to call that method with some paramters and get the HomeConsumptionConnection type back!

```
consumption = home.consumption(resolution = "HOURLY", last = 24)  # last 24 hours
print(consumption)  # <HomeConsumptionConnection: HomeConsumptionConnection>
```

Clicking the yellow underlined text "HomeConsumptionConnection" we see that it has a "nodes" attribute wich has a list of consumptions. Within the Consumption type you have all the goodies such as total cost, unit price, currency etc. Let's print the total cost of the last 24 hours!

```
running_total = 0
for node in consumption.nodes:
   running_total += node.total_cost  # Note that the naming convention for the␣
↪attributes is snake_case

print(running_total)  # 123.45
```

Looking back at the documentation, we see that the HomeConsumptionConnection type also has a PageInfo type. The PageInfo actually has a totalCost property that we can use instead of looping through all the nodes! Here's how to achieve the same thing as above, but using the page info we have instead!

```
print(consumption.page_info.total_cost)  # 123.45
```

Getting production information is very similar to getting consumption information. The only difference is that you use the "production" method instead of the "consumption" method. The rest is the same!

CHAPTER

FOUR

SENDING A PUSH NOTIFICATION

Sending a push notification is very simple. This sends a push notification to all your devices that are logged in to the Tibber app with the same account as the one you have generated your access token with.

```
import tibber

account = tibber.Account("your token")
account.send_push_notification("My title", "Hello! I'm a message!")
```

CHAPTER

FIVE

LIVE MEASUREMENTS

To get live measurements, you first have to register callback functions for the *live_measurement* event. This event is emitted every time a measurement has been made and has been retrieved from the API.

In simpler terms; in order to get live data, you need to create a function that you want to be run every time a live measurement is available. Then you must "register" that function so that it actually runs every time a live measurement is available.

Note: The live measurement may be delayed with a few seconds and is updated only every 2-10 seconds (in my experience).

```
import tibber

account = tibber.Account("your token")
home = account.homes[0]

@home.event("live_measurement")  # register the following function to run when the live_
↪measurement event is emitted
async def process_data(data):  # Note the data parameter in the function. This is␣
↪required and is of type LiveMeasurement.
   print(data.power)

# Now start retrieving live measurements
home.start_live_feed()
```

Note: Any code after home.start_live_feed() will not run! This is because the start_live_feed() method is blocking. It will run forever and will only stop when stopped with Ctrl+C or when the interpreter closes.

To close the live feed after any condition, you can pass the exit_condition argument to the start_live_feed() method. If the exit_condition function returns true, the live feed will be stopped (and code execution will continue).

```
import tibber

account = tibber.Account("your token")
home = account.homes[0]

@home.event("live_measurement")  # register the following function to run when the live_
↪measurement event is emitted
async def process_data(data):  # Note the data parameter in the function. This is␣
```

(continues on next page)

(continued from previous page)

```
↪required and is of type LiveMeasurement.
   print(data.power)

# Now start retrieving live measurements
home.start_live_feed(exit_condition = lambda: True)  # This will stop the live feed␣
↪after the first measurement
```

```
import tibber

account = tibber.Account("your token")
home = account.homes[0]

@home.event("live_measurement")  # register the following function to run when the live_
↪measurement event is emitted
async def process_data(data):  # Note the data parameter in the function. This is␣
↪required and is of type LiveMeasurement.
   print(data.power)

def my_exit_function(live_measurement_data):
   return live_measurement_data.power > 1000:

# Now start retrieving live measurements
home.start_live_feed(exit_condition = my_exit_function)  # This will stop the live feed␣
↪when the power is above 1000
print("We made it! The power is above 1000!")
```

CHAPTER

SIX

WELCOME TO TIBBER.PY'S DOCUMENTATION!

Note: This documentation is still in development and therefore does not cover all the functionality of the library. For now, you may read the Tibber API docs on the Tibber API explorer website to see all the properties and methods of each type in this library (this close correlation to the Tibber API is explained further in the Introduction section).

tibber.py is an unofficial wrapper library for the Tibber API developed and maintained by BeatsuDev. To quickly get started with common operations with the tibber.py library, you can head over to the GitHub repo and read the README.md file. There you can also see the source code of the library.

6.1 Introduction

Tibber is a relatively new electrical energy provider based in Norway (though they are available in several other nordic countries). They focus on providing useful gadgets and functionality to save energy consumption costs. For example, they provide functionality to charge your electric car when the price for elecricity is low. To expand on this feature, and perhaps add this to other smart devices in your home, you might consider using the Tibber API.

This is where tibber.py comes in to play!

With tibber.py you can retrieve data from the Tibber API super simple! This library aims to have 100% API coverage, meaning that whatever you can do with the Tibber API, you can do through this library too!

This API wrapper library is developed to be as close to the Tibber API in structure as possible. E.g. if a home has the property *consumption*, then this library's *Home* class will have a *consumption* property too. This way, besides from this documentation, you can read the Tibber API documentation and expect the library to work the same way.

6.2 How to use this documentation

If you are eager to get started, I would recommend reading some examples to see how things are done with this library. Once you've got a feel for how it is syntactically, you can go over to reading the API Reference (which is under development as of now, meanwhile you can use the Tibber API explorer). The API Reference covers *all* classes and objects you will find in this library, and lists all it's properties and methods.

6.3 How to contribute or report an issue

To report an issue or bug, go to the GitHub repository and create a new issue describing your problem and how to reproduce it. Also include what the *expected* behaviour was, and what the actual behaviour was.

> **Warning:** Issues that are created which do not include the problem, reproduction of the problem, the expected behaviour **and** the actual behaviour, will be tagged as invalid and will not be fixed or attended to.