PyPump Documentation

Release 0.7

Jessica Tallon

July 26, 2016

User documentation

Release v0.7.

PyPump is a simple but powerful and pythonic way of interfacing with the pump.io API.

PyPump is under the GPLv3. You should probably look to see if/how this may impact you and your programs if you wish to use PyPump. The community for PyPump lives on IRC in the #pypump channel on MegNet.

CHAPTER 1

Getting started

If you're new to PyPump and feeling a bit overwhelmed the Tutorial is the best place to go, however if you're familiar with python and understand pump, check out the Quick 'n Dirty! guide!

1.1 Installation

1.1.1 Using pip

The best way to install PyPump is via pip, if you haven't, setup:

```
$ virtualenv path/to/virtualenv
$ source path/to/virtualenv/bin/activate
$ pip install pypump
```

If you get an error which looks like:

```
Could not find a version that satisfies the requirement pypump (from versions: 0.1.6a, 0.1.7a, 0.1.8a
Cleaning up...
No distributions matching the version for pypump
```

You need to specify the latest version, for example:

```
$ pip install pypump==0.6
```

1.1.2 Using git

Warning: The code on git may break, the code on pip is likely to be much more stable

You can if you want the latest and greatest use the copy on git, to do this execute:

```
$ git clone https://github.com/xray7224/PyPump.git
$ cd PyPump
$ virtualenv .vt_env && . .vt_env/bin/activate
$ python setup.py develop
```

To keep this up to date use the following command inside the PyPump folder:

```
$ git pull
```

1.2 Quick ‘n Dirty!

1.2.1 Introduction

Warning: This is not complete, this is used as a fast intro for those fairly familiar or a reference for those who are a little rusty with PyPump

This guide is designed to get you up to speed and using this library in a very short amount of time, to do that I avoid long winded explanations, if you’re completely new to PyPump and/or pump.io please use Tutorial.

1.2.2 Getting connected

So we need to get started:

```
>>> from pypump import PyPump, Client
```

As Part of our application we will need to ask the user to input a verification code from the website to give us access as part of the OAuth mechanism, the function needs to take a URL and have the user allow our application, for example:

```
>>> def simple_verifier(url):
...     print('Please follow the instructions at the following URL:')
...     print(url)
...     return raw_input("Verifier: ") # the verifier is a string
```

First we must tell the server about ourselves:

```
>>> client = Client(
    webfinger="me@my.server.tld",
    name="Quick 'n dirty",
    type="native" # can be "native" or "web"
    )
```

Now make PyPump (the class for talking to pump):

```
>>> pump = PyPump(client=client, verifier_callback=simple_verifier)
```

Super, next, I wanna see my inbox:

```
>>> my_inbox = pump.me.inbox
>>> for activity in my_inbox[:20]:
...     print(activity)
```

Note: iterating over the inbox without any slice will iterate until the very first note in your inbox/feed/outbox

Oh, my friend Evan isn’t there, I probably need to follow him:

```
>>> evan = pump.Person("evan@e14n.org")
>>> evan.follow()
```

Awesome. Lets check again:

```
>>> for activity in my_inbox[:20]:
...     print(activity)
```

Evan likes PyPump, super!:

```
>>> activity = my_inbox[1] # second activity in my inbox
>>> awesome_note = activity.obj
>>> awesome_note.content
'Oh wow, PyPump is awesome!'
>>> awesome_note.like()
```

I wonder if someone else has liked that:

```
>>> awesome_note.likes
[me@my.server.org, joar@some.other.server]
```

Cool! Lets tell them about these docs:

```
>>> my_comment = pump.Comment("Guys, if you like PyPump check out the docs!")
>>> awesome_note.comment(my_comment)
```

I wonder what was posted last:

```
>>> latest_activity = my_inbox[0]
>>> print(latest_activity)
<Activity: jrobb posted an image>
```

Oh it's an image, lets see the thumbnail:

```
>>> url = latest_activity.obj.thumbnail.url
>>> fout = open("some_image.{0}".format(url.split(".")[-1]), "wb")
>>> import urllib2 # this will be different with python3
>>> fout.write(urllib2.urlopen(url).read())
>>> fout.close()
```

Hmm, I want to see a bigger version:

```
>>> large_url = latest_activity.obj.original.url
>>> print(large_url)
<Image at https://some.server/uploads/somefriend/2013/7/7/JkdX2.png">
>>> # you will find Images often hold other pump.Image objects, we just need to extra the url
>>> large_url = large_url.url
>>> fout = open("some_image_larger.{0}".format(large_url.split(".")[-1]), "wb")
>>> fout.write(urllib2.urlopen(url).read())
>>> fout.close()
```

That looks awesome, lets post a comment:

```
>>> my_comment = pump.Comment("Great, super imaeg")
>>> latest_activity.obj.comment(my_comment)
```

Oh no, I made a typo:

```
>>> my_comment.delete()
>>> my_comment.content = "Great, super image")
>>> latest_activity.obj.comment(my_comment)
```

Much better! Lets make a note to tell people how easy this all is:

```
>>> my_note = pump.Note("My gawd... PyPump is super easy to get started with")
>>> my_note.send()
```

But hold on though, that only sent it to followers? What gives:

```
>>> awesome_pump = pump.Note("PyPump is really awesome!")
>>> awesome_pump.to = pump.Public
>>> awesome_pump.cc = (pump.me.followers, pump.Person("MyFriend@server.com"))
>>> awesome_pump.send()
```

Oh cool that's sent to all my friends, So can i make my own lists:

```
>>> for my_list in pump.me.lists:
...     print(my_list)
Coworkers
Family
Friends
```

Oh are all those my lists that are defined. How do I send a note to them?:

```
>>> new_note = pump.Note("Work sucks!")
>>> new_note.to = pump.me.lists["Coworkers"]
>>> new_note.cc = pump.me.lists["Friends"]
```

So, can i send something to all of of the groups I made? Yep:

```
>>> another_note = pump.Note("This really goes to everyone in my groups?")
>>> another_note.to = list(pump.me.lists)
>>> another_note.cc = (pump.Person("moggers87@microca.st"), pump.Person("cwebber@identi.ca"))
>>> another_note.send()
```

Don't forget is there are any issues please issue them on our GitHub!

1.3 Tutorial

1.3.1 PyPump and the Pump API

PyPump is aiming to implement and interface with the Pump API, which is a federation protocol for the web. You can read the actual Pump API docs to get a sense of all that, but here's a high level overview.

The Pump API is all about ActivityStreams and sending JSON-encoded descriptions of activities back and forth across different users on different sites. At the highest conceptual level, it's not too different from the idea of email servers sending emails back and forth, but the messages (activities here) are much more specific and carry more specific meaning about what "type" of message is being sent back and forth. An activity can be a user "favoriting" something or "posting an image" or what have you.

In the world of email, each user has an email address; in the world of Pump, each user has a webfinger address. It looks pretty similar, but it's meant for the web. For the sake of this tutorial, you don't need to know how webfinger works; the PyPump will handle that for you.

Each user has two main feeds that are used for communication. In the Pump API docs' own wording:

- An **activity outbox** (probably at /api/user/<nickname>/feed). This is where the user posts new activities, and where others can read the user's activities.
- An **activity inbox** (probably at /api/user/<nickname>/inbox). This is where the user can read posts that were sent to him/her. Remote servers can post activities here to be delivered to the user.

(We use the inbox/outbox convention fairly strongly in PyPump.)

You can read the Pump spec, but sometimes coding examples are the best way to learn. So, that said, let's get into an example of using PyPump!

1.3.2 A quick example

Let's assume you already have a user with the webfinger id of mizbunny@example.org. We want to check what our latest messages are! But before we can do that, we need to authenticate. If this is your first time, you need to authenticate this client:

```
>>> from pypump import PyPump, Client
>>> client = Client(
...     webfinger="mizbunny@example.org",
...     type="native", # Can be "native" or "web"
...     name="Test.io"
...     )
>>> def simple_verifier(url):
...     print('Go to: ' + url)
...     return raw_input('Verifier: ') # they will get a code back
>>> pump = PyPump(client=client, verifier_callback=simple_verifier)
```

The PyPump call will try to verify with OAuth. You may wish to change how it asks for authentication. The `simple_verifier` function in the example above writes to standard out a URL for the user to click and reads in from standard in for a verification code presented by the webserver.

Note: By default PyPump will use the JSONStore which comes with PyPump. This will store the client and OAuth credentials created when you connect to pump at `~/$XDG_CONFIG_HOME/PyPump/credentials.json`. If you wish to change the path or store the data somewhere else (postgres, mongo, redis, etc.) we suggest you read the Store documentation.

Warning: You should store the client credentials and tokens somewhere safe, with this information anyone can access the user's pump.io account!

You can now reconnect like so:

```
>>> client = Client(
...     webfinger="mizbunny@example.org",
...     type="native",
...     name="Test.io",
...)
>>> pump = PyPump(
...     client=client,
...     verifier_callback=simple_verifier
...)
```

Okay, we're connected! Next up, we want to check out what our last 30 items in our inbox are, but first we need to find ourselves:

```
>>> me = pump.Person("mizbunny@example.org")
>>> me.summary
>>> 'Hello and welcome to my summary'
```

That looks like us, now to find our inbox items. The inbox comes in three versions

- me.inbox.major is where major activities such as posted notes and images end up.
- me.inbox.minor is where minor activities such as likes and comments end up.
- me.inbox is a combination of both of the above.

We only want to see notes, so we use the major inbox. The inbox supports python-style index slicing:

```
>>> recent_activities = me.inbox.major[:30]  # get last 30 activities
```

We could print out each of the most recent activities like so:

```
>>> for activity in recent_activities:
>>>     print(activity)
<Activity: Evan Prodromou posted a note>
<Activity: jrobb posted a note>
<Activity: jpope posted a note>
<Activity: sazius posted a note>
...
```

Maybe we're just looking at our most recent message, and see it's from our friend Evan. It seems that he wants to invite us over for a dinner party:

```
>>> activity = recent_activities[0]
>>> activity
<Activity: Evan Prodromou posted a note>
>>> message = activity.obj
>>> message.author
<User evan@e14n.com>
>>> message.content
"Yo, want to come over to dinner?  We're making asparagus!"
```

We can comment on the message saying we'd love to:

```
>>> our_reply = pump.Comment("I'd love to!")
>>> message.comment(our_reply)  # this is Evan's message we got above!
```

(Since this Note activity is being instantiated, it needs a reference to our PyPump class instance. Objects that you get back and forth from the API themselves will try to keep track of their own parent PyPump object for you.)

We could even like/favourite the previous message:

```
>>> message.like()
```

We can also check to see what our buddy's public feed is. Maybe he's said some interesting things?:

```
>>> evan = message.author
>>> for activity in evan.outbox:
>>>     message = activity.obj
>>>     print(message.content)
```

Perhaps we want to know a bit about Evan:

```
>>> print(evan.summary)
```

Maybe we took a picture, and we want to post that picture to our public feed so everyone can see it. We can do this by posting it to our outbox:

```
>>> img = pump.Image(
...     display_name="Sunset",
...     content="I took this the other day, came out really well!")
>>> img.from_file("sunset.jpg")
```

When posting an image or a note you may wish to post it to more people than just your followers (which is the default on most pump servers). You can easily do this by doing:

```
>>> my_note = pump.Note("This will go to everyone!")
>>> my_note.to = pump.Public
>>> my_note.send()
```

You can also send notes to specific people so if I wanted to send a note only to evan to invite him over, I could do something like this:

```
>>> my_note = pump.Note("Hey evan, would you like to come over later to check out PyPump")
>>> my_note.to = pump.Person("e14n@e14n.org")
>>> my_note.send() # Only evan will see this.
```

1.4 Authorization

1.4.1 What you need to know

Pump.io uses OAuth 1.0 with dynamic client registration, this is available through a lot of libraries, PyPump uses oauthlib and a wrapper around it to provide an provide an interface with the requests library - *requests-oauthlib* <*https://github.com/requests/requests-oauthlib*>. All of that is handled by PyPump however there are some things to know.

OAuth works by exchanging pre-established client credentials and tokens, you however have to provide those each time you instantiate the PyPump object. You will have to provide a mechanism to store these so that you can you can provide them the next time.

Note: As of version 0.6 PyPump is storing credentials using an internal Store object.

1.4.2 Example

The following will create (for the first time) a connection to a pump.io server for the user Tsyesika@io.theperplexingpariah.co.uk for my client named "Test.io":

```
>>> from pypump import PyPump, Client
>>> client = Client(
...     webfinger="Tsyesika@io.theperplexingpariah.co.uk",
...     name="Test.io",
...     type="native"
...)
>>> def simple_verifier(url):
...     print('Go to: ' + url)
...     return raw_input('Verifier: ') # they will get a code back
>>> pump = PyPump(client=client, verifier_callback=simple_verifier)
```

An example of then connecting again (using the same variable names as above). This will produce a PyPump object which will use the same credentials as established above:

```
>>> client = Client(
...     webfinger="Tsyesika@io.theperplexingpariah.co.uk",
...     name="Test.io",
...     type="native",
...     )
>>> pump = PyPump(
...         client=client,
...         verifier_callback=simple_verifier
...         )
```

1.5 Getting Verifier

For OAuth to allow OOB (Out of band) applications to have access to an account, first we must provide a link and instructions for the user. Then we must provide a means of copying the verifier into an application and relaying it to the server with other tokens. The server will then provide us with the credentials that we can use.

You must write a method which takes a URL that the user needs to visit, provide some way for that user to input a string value (the verification), and then give that value to PyPump. This could be simply a case of printing the link and using raw_input/input to get the verifier or it could be a more complex function which redraws a GUI and opens a browser.

1.5.1 Simple verifier

The following is an example of a simple verifier which could be used for a CLI (command line interface) application. This method is actually the same function which is used to prompt the user to provide a verifier in the PyPump Shell:

```
def verifier(url):
    """ Asks for verification code for OAuth OOB """
    print("Please open and follow the instructions:")
    print(url)
    return raw_input("Verifier: ")
```

1.5.2 Callback

Having a function which is called and then returns the verification might be more difficult in GUI programs or other interfaces. We provide a callback mechanism that you can use. If the verifier function returns None then PyPump assumes you will be calling PyPump.verifier which takes the verifier as the argument.

1.5.3 Complex GUI example

As an attempt to avoid writing an example which is tied to one GUI library, I have made one up in order to demonstrate exactly what might be involved:

```
import webbrowser # it's a python module - really, check it out.

from pypump import Client, PyPump

class MyWindow(guilib.Window):

    def __init__(self, *args, **kwargs):

        # Write out to the screen telling them what we're doing
        self.draw("Please wait, loading...")

        # setup pypump object
        client = Client(
            webfinger='someone@server.com',
            type='native',
            name='An awesome GUI client'
        )

        self.pump = PyPump(
            client=client,
            verifier_callback=self.ask_for_verifier
        )


    def ask_for_verifier(self, url):
        """ Takes a URL, opens it and asks for a verifier """
        # Open the URL in a browser
        webbrowser.open(url)

        # Clear other stuff from window
        self.clear_window()

        # draw a text input box and a button to submit
        self.draw(guilib.InputBox('Verifier:', name='verifier'))
        self.draw(guilib.Button('Verify!', onclick=self.verifier))

    def verifier(self, verifier):
        """ When the button is clicked it sends the verifier here which we give to PyPump """
        self.pump.verifier(verifier)
```

If you return anything from your verifier callback, pypump will expect that to be the verifier code so unless you're actually using a simple method, ensure you return None.

1.6 Web Development using PyPump

Warning: This section needs to be updated.

One of the problem with PyPump and Web development is that you often have a view which is called and then must return a function. While it is possible it may be difficult to use the regular PyPump callback routines. WebPump is a subclassed version of PyPump which handles that for you.

The only real difference is you don't specify a *verifier_callback* (if you do it will be ignored). Once the instanciation has completed you can guarantee that the URL for the callback has been created.

1.6.1 Django

This is an example of a very basic django view which uses WebPump:

```
from pypump import WebPump
from app.models import PumpModel
from django.shortcuts import redirect
from django.exceptions import ObjectDoesNotExist

def pump_view(request, webfinger):
    try:
        webfinger = PumpModel.objects.get(webfinger=webfinger)
    except ObjectDoesNotExist:
        webfinger = PumpModel.objects.create(webfinger=webfinger)
        webfinger.save()

    # make the WebPump object
    if webfinger.oauth_credentials:
        pump = WebPump(
                webfinger.webfinger,
                client_type="web",
                client_name="DjangoApp",
                key=webfinger.key,
                secret=webfinger.secret
                token=webfinger.token,
                token_secret=token_secret,
                callback_uri="http://my_app.com/oauth/authorize"
                )
    else:
        pump = WebPump(
                webfinger.webfinger,
                client_type="web",
                client_name="DjangoApp",
                callback_uri="http://my_app.com/oauth/authorize"
                )

    # save the client credentials as they won't change
    webfinger.key, webfinger.secret, webfinger.expirey = pump.get_registeration()

    # save the request tokens so we can identify the authorize callback
    webfinger.token, webfinger.secret = pump.get_registrat()

    # save the model back to db
    webfinger.save()

    if pump.url is not None:
        # The user must go to this url and will get bounced back to our
        # callback_uri we specified above and add the webfinger as a
        # session cookie.
        request.session["webfinger"] = webfinger
        return redirect(pump.url)

    # okay oauth completed successfully, we can just save the oauth
    # credentials and redirect.
    webfinger.token, webfinger.token_secret = pump.get_registration()
    webfinger.save()

    # redirect to profile!
    return redirect("/profile/{webfinger}".format(webfinger))

def authorize_view(request):
    """ This is the redirect when authorization is complete """
    webfinger = request.session.get("webfinger", None)
    token, verifier = request.GET["token"], request.GET["verifier"]

    try:
        webfinger = PumpModel.objects.get(
                webfiger=webfinger,
                token=token
                )

    except ObjectDoesNotExist:
        return redirect("/error") # tell them this is a invalid request

    pump = WebPump(
            webfinger.webfinger,
            client_name="DjangoApp",
            client_type="web",
            key=pump.key,
            secret=pump.secret,
            )

    pump.verifier(verifier)

    # Save the access tokens back now.
    webfinger.token, webfinger.token_secret = pump.get_registration()
    webfinger.save()

    # and redirect to their profile
    return redirect("/profile")
```

1.7 Store

Using store objects is the way PyPump handles the storing of certain data it gets that needs to be saved to disk. There is several pieces of data that might be stored such as:

- Client ID and secret
- OAuth request token and secret
- OAuth access token and secret

There might be others in the future too. The store object has an interface like a dictionary. PyPump provides a disk JSON store which allows you to easily just save the data to disk. You should note that this data should be considered sensative as with it someone has access to the users pump.io account.

1.7.1 Implementation

You probably want to provide your own storage object. There are two extra methods other than dictionary methods you need to implement are:

```
@classmethod
def load(cls, webfinger, pump):
    """
    This should return an instance of the store object full of any
    data that has been saved. It's your responsibility to set the
    `prefix` attribute on the store object.

    webfinger: String containing webfinger of user.
    pump: PyPump object loading the store object.
    """
    store = cls()
    store.prefix = webfinger
    return store

def save(self):
    """
    This should save all the data to the storage.
    """
    pass
```

There is a *prefix* attribute will contain the webfinger of the user the data belongs to. All the data stored and loaded should relate to this webfinger.

The save is called frequently and multiple times. The AbstractStore class will call the save method everytime something is set/changed on the object.

1.7.2 PyPump

There are several ways to provide PyPump with a store object. You can pass it in when you create the PyPump object e.g:

```
>>> my_store = MyStore.load()
>>> pump = PyPump(store=my_store, ...)
```

If no storage object is passed, PyPump will call the .create_store method on itself. This will by default call .load(webfinger, pypump) on whatever class is in store_class on PyPump. You can provide your own class there:

```
>>> class MyPump(PyPump):
...     store_class = MyStore
...
>>> pump = MyPump(...)
```

This will use the MyStore class. If you want to do something else you can always override the .create_store method:

```
class MyPump(PyPump):
    def create_store(self):
        """ This should create and return the store object """
        return MyStore.load(
            webfinger=self.client.webfinger,
            pump=self
        )
```

For convenience, PyPump comes with a simple JSON store class, *pypump.store.Store*.

1.8 Upload Media

Uploading images (and in the future other types of media) via PyPump is relatively easy. Firstly you will need to setup a Client and PyPump model, information for how to do that can be found in the quick and dirty guide.

Once you've uploaded an image you will be able to do:

```
>>> my_image = pump.Image()
>>> my_image.from_file("/home/jessica/my_image.jpg")
```

That is enough to post an image to pump (or MediaGoblin). You can now look at the URL on the image model:

```
>>> my_image.url
'https://microca.st/Tsyesika/image/yZJCA42GTfCuaeEBqyc26Q'
```

1.8.1 Interacting with Media

Commenting

You can then interact with this image:

```
>> my_image.comment("Hai, this si my comment")
```

Liking

If you want to like some media you can use:

```
>> my_image.like()
```

Note: MediaGoblin currently doesn't support liking.

Deleting

Deleting media is also easy:

```
>> my_image.delete()
```

Note: MediaGoblin current doesn't support deletion of media.

1.9 Class reference

1.9.1 Essentials

Classes doing most of the work.

class `pypump.`**`PyPump`**(*client*, *verifier_callback*, *store=None*, *callback='oob'*, *verify_requests=True*, *retries=0*, *timeout=30*)

Main class to interface with PyPump.

This class keeps everything together and is responsible for making requests to the server on it's own behalf and on the behalf of the other clients as well as handling the OAuth requests.

Parameters

- **`client`** – an instance of `Client`.
- **`verifier_callback`** – If this is our first time registering the client, this function will be called with a single argument, the url one can post to for completing verification.
- **`store`** – this is the `pypump.Store` instance to save any data persistantly.
- **`callback`** – the URI that is used for redirecting a user after they authenticate this client... assuming this is happening over the web. If not, the callback is "oob", or "out of band".
- **`verify_requests`** – If this is set to False PyPump won't check SSL/TLS certificates.
- **`retries`** – number of times to retry if a request fails.
- **`timeout`** – how long to give on a timeout for an http request, in seconds.

class `pypump.`**`Client`**(*webfinger*, *type*, *name=None*, *contacts=None*, *redirect=None*, *logo=None*, *key=None*, *secret=None*, *expirey=None*)

This represents a client/application which is using the Pump API.

Parameters

- **`webfinger`** – webfinger id of this user, ie "mary@example.org" This is probably your username @ the domain of the pump instance you're using.

- **type** – whether this is a “web” or “native” client. Unless you’re using pypump as part of a web service, you should choose “native”.
- **name** – The name of this client, ie your application’s name. For example, if you were using PyPump to write CoolCommunicator, you would put “CoolCommunicator” here.
- **contacts** – Who wrote this application? List of email addresses of those who authored this client.
- **redirect** – a list of URIs as callbacks for the authorization server
- **logo** – URI of your application’s logo.

Additionally, the following init args should only be supplied if you’ve already registered this client with the server. If not, these will be filled in during the *self.register()* step.

Parameters

- **key** – If This is the token (the *client_id*) we got back that identifies our client, assuming we’ve already registered our client.
- **secret** – This is your secret token that authorizes you to connect to the pump instance (the *client_secret*).
- **expirey** – When our token expires (*espires_at*)

Note that the above three are not the same as the oauth permissions verifying the user has access to post, this is related to permissions and identification of the client software to post. For the premission related to the access of the account, see PyPump.key and PyPump.secret.

1.9.2 Pump objects

Classes representing pump.io objects:

- *Note*
- *Image*
- *Audio*
- *Video*
- *Comment*
- *Person*
- *Inbox*
- *Outbox*
- *Lists*

class pypump.models.note.**Note**(*content=None*, *display_name=None*, ***kwargs*)

This object represents a pump.io **note**, notes are used to post text (or html) messages to the pump.io network.

Parameters

- **content** – (optional) Note content.
- **display_name** – (optional) Note title.

Usage:

```
>>> mynote = pump.Note(content='<b>Hello</b> world!')
>>> mynote.send()
```

cc

List of secondary recipients. The object will show up in the recipients inbox when sent.

Example:

```
>>> mynote = pump.Note('hello world')
>>> mynote.cc = pump.Public
```

comment (*comment*)

Add a *Comment* to the object.

Parameters **comment** – A *Comment* instance, text content is also accepted.

Example:

```
>>> anote.comment(pump.Comment('I agree!'))
```

comments

A *Feed* of the comments for the object.

Example:

```
>>> for comment in mynote.comments:
...     print(comment)
...
comment by pypumptest2@pumpyourself.com
```

delete ()

Delete the object content on the server.

Example:

```
>>> mynote.deleted
>>> mynote.delete()
>>> mynote.deleted
datetime.datetime(2014, 10, 19, 9, 26, 39, tzinfo=tzutc())
```

favorite ()

Favorite the object.

favorites

A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

like ()

Like the object.

Example:

```
>>> anote.liked
False
>>> anote.like()
>>> anote.liked
True
```

likes

A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

send()

Send the object to the server.

Example:

```
>>> mynote = pump.Note('Hello world!)
>>> mynote.send()
```

share()

Share the object.

Example:

```
>>> anote.share()
```

shares

A *Feed* of the people who've shared the object.

Example:

```
>>> for person in mynote.shares:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

to

List of primary recipients. If entry is a `Person` the object will show up in their direct inbox when sent.

Example:

```
>>> mynote = pump.Note('hello pypumptest1')
>>> mynote.to = pump.Person('pypumptest1@pumpity.net')
>>> mynote.to
[<Person: pypumptest1@pumpity.net>]
```

unfavorite()

Unfavorite a previously favorited object.

unlike()

Unlike a previously liked object.

Example:

```
>>> anote.liked
True
>>> anote.unlike()
>>> anote.liked
False
```

unshare()
Unshare a previously shared object.

Example:

```
>>> anote.unshare()
```

class pypump.models.media.**Image**(*display_name=None*, *content=None*, *license=None*, ***kwargs*)
This object represents a pump.io **image**, images are used to post image content with optional text (or html) messages to the pump.io network.

Parameters

- **content** – (optional) Image text content.
- **display_name** – (optional) Image title.

Example:

```
>>> myimage = pump.Image(display_name='Happy Caturday!')
>>> myimage.from_file('/path/to/kitteh.png')
```

thumbnail
ImageContainer holding information about the thumbnail image.

original
ImageContainer holding information about the original image.

cc
List of secondary recipients. The object will show up in the recipients inbox when sent.

Example:

```
>>> mynote = pump.Note('hello world')
>>> mynote.cc = pump.Public
```

comment(*comment*)
Add a *Comment* to the object.

Parameters **comment** – A *Comment* instance, text content is also accepted.

Example:

```
>>> anote.comment(pump.Comment('I agree!'))
```

comments
A *Feed* of the comments for the object.

Example:

```
>>> for comment in mynote.comments:
...     print(comment)
...
comment by pypumptest2@pumpyourself.com
```

delete()
Delete the object content on the server.

Example:

```
>>> mynote.deleted
>>> mynote.delete()
>>> mynote.deleted
datetime.datetime(2014, 10, 19, 9, 26, 39, tzinfo=tzutc())
```

favorite()
Favorite the object.

favorites
A `Feed` of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

from_file(*filename*)
Uploads a file from a filename on your system.

Parameters **filename** – Path to file on your system.

Example:

```
>>> myimage.from_file('/path/to/dinner.png')
```

like()
Like the object.

Example:

```
>>> anote.liked
False
>>> anote.like()
>>> anote.liked
True
```

likes
A `Feed` of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

share()

Share the object.

Example:

```
>>> anote.share()
```

shares

A *Feed* of the people who've shared the object.

Example:

```
>>> for person in mynote.shares:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

to

List of primary recipients. If entry is a `Person` the object will show up in their direct inbox when sent.

Example:

```
>>> mynote = pump.Note('hello pypumptest1')
>>> mynote.to = pump.Person('pypumptest1@pumpity.net')
>>> mynote.to
[<Person: pypumptest1@pumpity.net>]
```

unfavorite()

Unfavorite a previously favorited object.

unlike()

Unlike a previously liked object.

Example:

```
>>> anote.liked
True
>>> anote.unlike()
>>> anote.liked
False
```

unshare()

Unshare a previously shared object.

Example:

```
>>> anote.unshare()
```

class `pypump.models.media.`**Audio**(*display_name=None*, *content=None*, *license=None*, ***kwargs*)

This object represents a pump.io **audio** object, audio objects are used to post audio content with optional text (or html) messages to the pump.io network.

Parameters

- **content** – (optional) Audio text content.
- **display_name** – (optional) Audio title.

Example:

```
>>> myogg = pump.Audio(display_name='Happy Caturday!')
>>> myogg.from_file('/path/to/kitteh.ogg')
```

stream
StreamContainer holding information about the stream.

cc
List of secondary recipients. The object will show up in the recipients inbox when sent.

Example:

```
>>> mynote = pump.Note('hello world')
>>> mynote.cc = pump.Public
```

comment (*comment*)
Add a *Comment* to the object.

Parameters **comment** – A *Comment* instance, text content is also accepted.

Example:

```
>>> anote.comment(pump.Comment('I agree!'))
```

comments
A *Feed* of the comments for the object.

Example:

```
>>> for comment in mynote.comments:
...     print(comment)
...
comment by pypumptest2@pumpyourself.com
```

delete ()
Delete the object content on the server.

Example:

```
>>> mynote.deleted
>>> mynote.delete()
>>> mynote.deleted
datetime.datetime(2014, 10, 19, 9, 26, 39, tzinfo=tzutc())
```

favorite ()
Favorite the object.

favorites
A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

from_file (*filename*)

Uploads a file from a filename on your system.

Parameters filename – Path to file on your system.

Example:

```
>>> myimage.from_file('/path/to/dinner.png')
```

like ()

Like the object.

Example:

```
>>> anote.liked
False
>>> anote.like()
>>> anote.liked
True
```

likes

A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

share ()

Share the object.

Example:

```
>>> anote.share()
```

shares

A *Feed* of the people who've shared the object.

Example:

```
>>> for person in mynote.shares:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

to

List of primary recipients. If entry is a `Person` the object will show up in their direct inbox when sent.

Example:

```
>>> mynote = pump.Note('hello pypumptest1')
>>> mynote.to = pump.Person('pypumptest1@pumpity.net')
>>> mynote.to
[<Person: pypumptest1@pumpity.net>]
```

unfavorite()
Unfavorite a previously favorited object.

unlike()
Unlike a previously liked object.

Example:

```
>>> anote.liked
True
>>> anote.unlike()
>>> anote.liked
False
```

unshare()
Unshare a previously shared object.

Example:

```
>>> anote.unshare()
```

class `pypump.models.media.`**Video**(*display_name=None*, *content=None*, *license=None*, ***kwargs*)
This object represents a pump.io **video** object, video objects are used to post video content with optional text (or html) messages to the pump.io network.

Parameters

- **content** – (optional) Video text content.
- **display_name** – (optional) Video title.

Example:

```
>>> myogv = pump.Video(display_name='Happy Caturday!')
>>> myogv.from_file('/path/to/kitteh.ogv')
```

stream
StreamContainer holding information about the stream.

cc
List of secondary recipients. The object will show up in the recipients inbox when sent.

Example:

```
>>> mynote = pump.Note('hello world')
>>> mynote.cc = pump.Public
```

comment(*comment*)
Add a *Comment* to the object.

Parameters comment – A *Comment* instance, text content is also accepted.

Example:

```
>>> anote.comment(pump.Comment('I agree!'))
```

comments

A *Feed* of the comments for the object.

Example:

```
>>> for comment in mynote.comments:
...     print(comment)
...
comment by pypumptest2@pumpyourself.com
```

delete()

Delete the object content on the server.

Example:

```
>>> mynote.deleted
>>> mynote.delete()
>>> mynote.deleted
datetime.datetime(2014, 10, 19, 9, 26, 39, tzinfo=tzutc())
```

favorite()

Favorite the object.

favorites

A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

from_file(*filename*)

Uploads a file from a filename on your system.

Parameters **filename** – Path to file on your system.

Example:

```
>>> myimage.from_file('/path/to/dinner.png')
```

like()

Like the object.

Example:

```
>>> anote.liked
False
>>> anote.like()
>>> anote.liked
True
```

likes
A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

share()
Share the object.

Example:

```
>>> anote.share()
```

shares
A *Feed* of the people who've shared the object.

Example:

```
>>> for person in mynote.shares:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

to
List of primary recipients. If entry is a `Person` the object will show up in their direct inbox when sent.

Example:

```
>>> mynote = pump.Note('hello pypumptest1')
>>> mynote.to = pump.Person('pypumptest1@pumpity.net')
>>> mynote.to
[<Person: pypumptest1@pumpity.net>]
```

unfavorite()
Unfavorite a previously favorited object.

unlike()
Unlike a previously liked object.

Example:

```
>>> anote.liked
True
>>> anote.unlike()
>>> anote.liked
False
```

unshare()
Unshare a previously shared object.

Example:

```
>>> anote.unshare()
```

class `pypump.models.comment.`**`Comment`**(*content=None*, *in_reply_to=None*, ***kwargs*)

This object represents a pump.io **comment**, comments are used to post text (or html) messages in reply to other objects on the pump.io network.

Parameters

- **`content`** – (optional) Comment content.
- **`in_reply_to`** – (optional) Object to reply to.

Example:

```
>>> catpic
<Image by alice@example.org>
>>> mycomment = pump.Comment(content='Best cat pic ever!', in_reply_to=catpic)
>>> mycomment.send()
```

`cc`

List of secondary recipients. The object will show up in the recipients inbox when sent.

Example:

```
>>> mynote = pump.Note('hello world')
>>> mynote.cc = pump.Public
```

`comment`(*comment*)

Add a *`Comment`* to the object.

Parameters **`comment`** – A *`Comment`* instance, text content is also accepted.

Example:

```
>>> anote.comment(pump.Comment('I agree!'))
```

`comments`

A *`Feed`* of the comments for the object.

Example:

```
>>> for comment in mynote.comments:
...     print(comment)
...
comment by pypumptest2@pumpyourself.com
```

`delete`()

Delete the object content on the server.

Example:

```
>>> mynote.deleted
>>> mynote.delete()
>>> mynote.deleted
datetime.datetime(2014, 10, 19, 9, 26, 39, tzinfo=tzutc())
```

`favorite`()

Favorite the object.

favorites
A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

like()
Like the object.

Example:

```
>>> anote.liked
False
>>> anote.like()
>>> anote.liked
True
```

likes
A *Feed* of the people who've liked the object.

Example:

```
>>> for person in mynote.likes:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

send()
Send the object to the server.

Example:

```
>>> mynote = pump.Note('Hello world!)
>>> mynote.send()
```

share()
Share the object.

Example:

```
>>> anote.share()
```

shares
A *Feed* of the people who've shared the object.

Example:

```
>>> for person in mynote.shares:
...     print(person.webfinger)
...
pypumptest1@pumpity.net
pypumptest2@pumpyourself.com
```

to

List of primary recipients. If entry is a `Person` the object will show up in their direct inbox when sent.

Example:

```
>>> mynote = pump.Note('hello pypumptest1')
>>> mynote.to = pump.Person('pypumptest1@pumpity.net')
>>> mynote.to
[<Person: pypumptest1@pumpity.net>]
```

unfavorite()

Unfavorite a previously favorited object.

unlike()

Unlike a previously liked object.

Example:

```
>>> anote.liked
True
>>> anote.unlike()
>>> anote.liked
False
```

unshare()

Unshare a previously shared object.

Example:

```
>>> anote.unshare()
```

class `pypump.models.person.`**Person**(*webfinger=None*, ***kwargs*)

This object represents a pump.io **person**, a person is a user on the pump.io network.

Parameters **webfinger** – User ID in `nickname@hostname` format.

Example:

```
>>> alice = pump.Person('alice@example.org')
>>> print(alice.summary)
Hi, I'm Alice
>>> mynote = pump.Note('Hey Alice, it's Bob!')
>>> mynote.to = alice
>>> mynote.send()
```

cc

List of secondary recipients. The object will show up in the recipients inbox when sent.

Example:

```
>>> mynote = pump.Note('hello world')
>>> mynote.cc = pump.Public
```

favorites

Feed with all objects liked/favorited by the person.

Example:

```
>>> for like in pump.me.favorites[:3]:
...     print(like)
...
note by alice@example.org
image by bob@example.org
comment by evan@e14n.com
```

follow()
Follow person

followers
Feed with all *Person* objects following the person.

Example:

```
>>> alice = pump.Person('alice@example.org')
>>> for follower in alice.followers[:2]:
...     print(follower.id)
...
acct:bob@example.org
acct:carol@example.org
```

following
Feed with all *Person* objects followed by the person.

Example:

```
>>> bob = pump.Person('bob@example.org')
>>> for followee in bob.following[:3]:
...     print(followee.id)
...
acct:alice@example.org
acct:duncan@example.org
```

inbox
Inbox feed with all `activities` received by the person, can only be read if logged in as the owner.

Example:

```
>>> for activity in pump.me.inbox[:2]:
...     print(activity.id)
...
https://microca.st/api/activity/BvqXQOwXShSey1HxYuJQBQ
https://pumpyourself.com/api/activity/iQGdnz5-T-auXnbUUdXh-A
```

lists
Lists feed with all lists owned by the person.

Example:

```
>>> for list in pump.me.lists:
...     print(list)
...
Acquaintances
Family
Coworkers
Friends
```

outbox

Outbox feed with all `activities` sent by the person.

Example:

```
>>> for activity in pump.me.outbox[:2]:
...     print(activity)
...
pypumptest2 unliked a comment in reply to a note
pypumptest2 deleted a note
```

to

List of primary recipients. If entry is a `Person` the object will show up in their direct inbox when sent.

Example:

```
>>> mynote = pump.Note('hello pypumptest1')
>>> mynote.to = pump.Person('pypumptest1@pumpity.net')
>>> mynote.to
[<Person: pypumptest1@pumpity.net>]
```

unfollow()

Unfollow person

update()

Updates person object

class pypump.models.feed.**Inbox**(**args*, ***kwargs*)

This object represents a pump.io **inbox feed**, it contains all activities posted to the owner of the inbox.

Example:

```
>>> for activity in pump.me.inbox.items(limit=3):
...     print(activity)
Alice posted a note
Bob posted a comment in reply to a note
Alice liked a comment
```

direct

Direct inbox feed, contains activities addressed directly to the owner of the inbox.

items(*offset=None*, *limit=20*, *since=None*, *before=None*, **args*, ***kwargs*)

Get a feed's items.

Parameters

- **offset** – Amount of items to skip before returning data
- **since** – Return items added after this id (ordered old -> new)
- **before** – Return items added before this id (ordered new -> old)
- **limit** – Amount of items to return

major

Major inbox feed, contains major activities such as notes and images.

minor

Minor inbox feed, contains minor activities such as likes, shares and follows.

class pypump.models.feed.**Outbox**(**args*, ***kwargs*)

This object represents a pump.io **outbox feed**, it contains all activities posted by the owner of the outbox.

Example:

```
>>> for activity in pump.me.outbox.items(limit=3):
...     print(activity)
Bob posted a note
Bob liked an image
Bob followed Alice
```

items (*offset=None*, *limit=20*, *since=None*, *before=None*, **args*, ***kwargs*)
Get a feed's items.

Parameters

- **offset** – Amount of items to skip before returning data
- **since** – Return items added after this id (ordered old -> new)
- **before** – Return items added before this id (ordered new -> old)
- **limit** – Amount of items to return

major
Major outbox feed, contains major activities such as notes and images.

minor
Minor outbox feed, contains minor activities such as likes, shares and follows.

class pypump.models.feed.**Lists** (*url=None*, **args*, ***kwargs*)
This object represents a pump.io **lists feed**, it contains the *collections* (or lists) created by the owner.

Example:

```
>>> for i in pump.me.lists.items():
...     print(i)
Coworkers
Acquaintances
Family
Friends
```

create (*display_name*, *content=None*)
Create a new user list *collection*.

Parameters

- **display_name** – List title.
- **content** – (optional) List description.

Example:

```
>>> pump.me.lists.create(display_name='Friends', content='List of friends')
>>> myfriends = pump.me.lists['Friends']
>>> print(myfriends)
Friends
```

items (*offset=None*, *limit=20*, *since=None*, *before=None*, **args*, ***kwargs*)
Get a feed's items.

Parameters

- **offset** – Amount of items to skip before returning data
- **since** – Return items added after this id (ordered old -> new)

- **before** – Return items added before this id (ordered new -> old)
- **limit** – Amount of items to return

1.9.3 Low level objects

Classes you probably don't need to know about.

class pypump.models.media.**ImageContainer**(*url*, *width*, *height*)
Container that holds information about an image.

Parameters

- **url** – URL to image file on the pump.io server.
- **width** – Width of the image.
- **height** – Height of the image.

class pypump.models.media.**StreamContainer**(*url*)
Container that holds information about a stream.

Parameters **url** – URL to the file on the pump.io server.

class pypump.models.feed.**Feed**(*url=None*, **args*, ***kwargs*)
This object represents a basic pump.io **feed**, which is used for navigating a list of objects (inbox, followers, shares, likes and so on).

items(*offset=None*, *limit=20*, *since=None*, *before=None*, **args*, ***kwargs*)
Get a feed's items.

Parameters

- **offset** – Amount of items to skip before returning data
- **since** – Return items added after this id (ordered old -> new)
- **before** – Return items added before this id (ordered new -> old)
- **limit** – Amount of items to return

class pypump.models.collection.**Collection**(*id=None*, ***kwargs*)
This object represents a pump.io **collection**, collections have a members *Feed* and methods for adding and removing objects to that feed.

Parameters **id** – (optional) Collection id.

Example:

```
>>> friendlist = pump.me.lists['Friends']
>>> list(friendlist.members)
[<Person: alice@example.org>]
>>> friendlist.add(pump.Person('bob@example.org'))
```

add(*obj*)
Adds a member to the collection.

Parameters **obj** – Object to add.

Example:

```
>>> mycollection.add(pump.Person('bob@example.org'))
```

delete()

Delete the object content on the server.

Example:

```
>>> mynote.deleted
>>> mynote.delete()
>>> mynote.deleted
datetime.datetime(2014, 10, 19, 9, 26, 39, tzinfo=tzutc())
```

members

Feed of collection members.

remove(*obj*)

Removes a member from the collection.

Parameters **obj** – Object to remove.

Example:

```
>>> mycollection.remove(pump.Person('bob@example.org'))
```

CHAPTER 2

About PyPump

2.1 Changelog

2.1.1 0.7

- Fixed bug where Image.original never got any info #145
- Added Audio and Video objects
- Work around bug in pump.io favorites feed which only let us get 20 latest items #65
- Person.update() now updates Person.location
- Fixed bug where PyPump with Python3 failed to save credentials to theJSONStore
- Dropped Python 2.6 support, PyPump now supports Python 2.7 or 3.3+
- PyPump now tries HTTPS first, and then only falls back to HTTP if `verify_requests` is `False`
- Fixed bug where `PyPump.request()` didnt sign oauth request on redirect #120
- Implement list methods on ItemList and Feed (`__getitem__` and `__len__`)
- Moved exceptions into single module and removed some unused exceptions. Make sure to update your import statements!

2.1.2 0.6

- Person no longer accept a webfinger without a hostname
- PumpObject.add_link and .add_links renamed to ._add_link and ._add_links
- Recipients can now be set for Comment, Person objects
- Recipient properties (.to, .cc, .bto, .bcc) has been moved from Activity to Activity.obj
- Feeds (inbox, followers, etc) can now be sliced by object or object_id.
- `Feed.items(offset=int|since=id|before=id, limit=20)` method.
- Unicode improvements when printing Pump objects.
- Instead of skipping an Object attribute which has no response data (f.ex Note.deleted when note has not been deleted) we now set the attribute to None.
- Fixed WebPump OAuth token issue (#89)

- Allow you to use `<commentable object\>.comment()` by passing in just a string apposed to a Comment object. 44f3426
- Introduce "Store" object for saving persistant data.

2.1.3 Earlier Releases

Sorry we didn't keep a change log prior to 0.6, you'll have to fish through the git commits to see what's changed.

2.2 Links

- PyPump on Github
- PyPump on PyPI

This was build on July 24, 2016