imvu.py Documentation

Release 0.6.1a

Doctor Jew

May 05, 2018

Contents

1 Documentation Contents **3**
- 1.1 Introduction . . . 3
- 1.2 Quickstart . . . 4
- 1.3 Setting Up Logging . . . 5
- 1.4 API Reference . . . 6

2 Extensions **13**
- 2.1 `imvu.ext.outfit` – Outfit Parser and Viewer . . . 13

imvu.py is a an easy to use API wrapper for IMVU.

Features:

- Easy to use with an object oriented design
- Optimized for both speed and memory

CHAPTER 1

Documentation Contents

1.1 Introduction

This is the documentation for imvu.py, a library for Python to aid in creating applications that utilize the IMVU API.

1.1.1 Prerequisites

imvu.py works with Python 3.6 or higher. Support for earlier versions of Python is not provided. Python 2.7 or lower is not supported. Python versions below 3.6 are not supported due to the lack of `typing` and `f-string` syntax support.

1.1.2 Installing

You can get the library directly from PyPI:

```
python3 -m pip install imvu.py
```

If you are using Windows, then the following should be used instead:

```
py -3 pip install -U imvu.py
```

Remember to check your permissions!

Virtual Environments

Sometimes we don't want to pollute our system installs with a library or we want to maintain different versions of a library than the currently system installed one. Or we don't have permissions to install a library along side with the system installed ones. For this purpose, the standard library as of 3.3 comes with a concept called "Virtual Environment" to help maintain these separate versions.

A more in-depth tutorial is found on the official documentation.

However, for the quick and dirty:

1. Go to your project's working directory:

```
$ cd your-script-source
$ python3 -m venv script-env
```

2. Activate the virtual environment:

```
$ source script-env/bin/activate
```

On Windows you activate it with:

```
$ bot-env\Scripts\activate.bat
```

3. Use pip like usual:

```
$ pip install -U imvu.py
```

Congratulations. You now have a virtual environment all set up without messing with your system installation.

1.2 Quickstart

This page gives a brief introduction to the library. It assumes you have the library installed, if you don't check out the *Installing* portion.

1.2.1 A Minimal Example

Let's make a script that will download and save a product.

It looks something like this:

```
import imvu
import asyncio


async def main():
    client = imvu.Client(username='Test', password='124512')

    product = await client.get_product(12345678)
    await product.download()


if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
    loop.close()
```

Name this file whatever you want, `product_download.py` for example. Make sure not to name it `imvu.py` because that will conflict with the library.

While there isn't too much going on here, here is a breakdown step by step.

1. The first line imports the library, if this raises a *ModuleNotFoundError* or *ImportError* then head on over to *Installing* section to properly install.
2. The second line imports *asyncio*, this library is needed for asynchronous execution.

3. We create an asynchronous function called *main()*.
4. Next, we create an instance of a *Client*. This client is our connection to IMVU.
5. Then we get a product by awaiting *Client.get_product()* and assigning it to the variable *product*
6. After we await the *Product.download()* method and pass in the location where we want to save the data to.

Now that we've made a script, we have to *run* the script. Luckily, this is simple since this is just a Python script, we can run it directly.

On Windows:

```
$ py -3 product_download.py
```

On other systems:

```
$ python3 product_download.py
```

Now you can try playing around with your basic script.

1.3 Setting Up Logging

imvu.py logs errors and debug information via the `logging` python module. It is strongly recommended that the logging module is configured, as no errors or warnings will be output if it is not set up. Configuration of the `logging` module can be as simple as:

```
import logging

logging.basicConfig(level=logging.INFO)
```

Placed at the start of the application. This will output the logs from discord as well as other libraries that uses the `logging` module directly to the console.

The optional `level` argument specifies what level of events to log out and can any of `CRITICAL`, `ERROR`, `WARNING`, `INFO`, and `DEBUG` and if not specified defaults to `WARNING`.

More advanced setups are possible with the `logging` module. To for example write the logs to a file called `imvu.log` instead of outputting them to to the console the following snippet can be used:

```
import imvu
import logging

logger = logging.getLogger('imvu')
logger.setLevel(logging.DEBUG)
handler = logging.FileHandler(filename='imvu.log', encoding='utf-8', mode='w')
handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:%(name)s:
↪%(message)s'))
logger.addHandler(handler)
```

This is recommended, especially at verbose levels such as `INFO`, and `DEBUG` as there are a lot of events logged and it would clog the stdout of your program.

If you want to take things a step further, you could log to both a file and to the console. To do this the following snippet can be used:

```
import imvu
import logging
from logging.handlers import RotatingFileHandler

logger = logging.getLogger('imvu')
logger.setLevel(logging.INFO)
log_format = logging.Formatter('%(asctime)s:%(levelname)s:%(name)s: %(message)s')

ch = logging.StreamHandler(sys.stdout)
ch.setFormatter(log_format)
ch.setLevel(logging.INFO)
logger.addHandler(ch)

fh = RotatingFileHandler(filename='imvu.log', maxBytes=1024 * 5, backupCount=2,␣
↪encoding='utf-8', mode='w')
fh.setFormatter(log_format)
logger.addHandler(fh)
```

For more information, check the documentation and tutorial of the `logging` module.

1.4 API Reference

The following section outlines the API of imvu.py.

Note: This module uses the Python logging module to log diagnostic and errors in an output independent way. If the logging module is not configured, these logs will not be output anywhere. See *Setting Up Logging* for more information on how to set up and use the logging module with imvu.py.

1.4.1 Version Related Info

There are two main ways to query version information about the library.

`imvu.`**`version_info`**
: A named tuple that is similar to sys.version_info.

  Just like sys.version_info the valid values for `release_level` are 'alpha', 'beta', 'candidate' and 'final'.

`imvu.`**`__version__`**
: A string representation of the version. e.g. `'3.1.0a'`.

1.4.2 Client

`class` `imvu.`**`Client`**(*username=None*, *password=None*, *loop=None*)
: Represents a client connection that connects to IMVU. This class is used to interact with the IMVU API.

  Parameters

  - **`username`** – Login username to IMVU.
  - **`password`** – Login password to IMVU.
  - **`loop`** – Optional event loop to use.

get_avatar(*id*) → typing.Union[imvu.avatar.Avatar, NoneType]
Lookup and retrieve an avatar.

Parameters **id** – The ID of the avatar to retrieve.

Returns A *Avatar* if found, else None.

get_product(*id*, *hidden=False*) → typing.Union[imvu.product.Product, NoneType]
Lookup and retrieve a product.

Parameters

- **id** – The ID of the product to retrieve.
- **hidden** – Attempt to retrieve the product even if hidden.

Returns A *Product* if found, else None.

get_raw(*path*, *id: int = 0*) → typing.Union[dict, NoneType]
Returns the raw JSON data for a given `Route` and ID.

Parameters

- **path** (*str*) – API path to lookup.
- **id** (*int*) – ID to inject into the API path.

Returns Raw dictionary of data retrieved. If not found, returns None.

Return type dict or None

get_user(*id: int = None*, *username: str = None*) → typing.Union[imvu.user.User, NoneType]
Lookup and retrieve a user

Parameters

- **id** – The ID of the user to retrieve.
- **username** – Username to lookup and retrieve.

Returns A *User* if found, else None.

1.4.3 Enumerations

The API provides some enumerations for certain types of strings to avoid the API from being stringly typed in case the strings change in the future.

All enumerations are subclasses of enum.

class imvu.**Availability**
Specifies the availability of a *User*.

ONLINE
User is online.

AVAILABLE
User is online and available.

AWAY
User is online but away.

DND
User is online but chooses to ignore notifications.

DO_NOT_DISTURB
An alias for *DND*.

OFFLINE
User is completely offline.

class imvu.**Visibility**
Specifies the visibility of a *Product*.

VISIBLE
Product is visible from the shop and purchasable.

DISPLAY_ONLY
Product is visible from the shop but not purchasable.

HIDDEN
Product is not visible in the shop and may only be obtained through the creator.

REMOVED
Product is hidden but cannot be obtained through any means.

class imvu.**Gender**
Specifies the gender of a *User* or *Product*.

MALE
Male sex.

FEMALE
Female sex.

HIDDEN
Gender is hidden or unavailable.

class imvu.**RelationshipStatus**
Specifies a *User*'s relationship status.

PREFER_NOT_TO_SAY
User declined to state their relationship status.

SINGLE
User is single.

SEEING_SOMEONE
User is seeing someone.

RELATIONSHIP
User is in a relationship.

MARRIED
User is married.

DIVORCED
User is divorced (was married).

OTHER
Similar to prefer not to say, but indecisive.

class imvu.**Orientation**
Specifies a *User* 's orientation.

PREFER_NOT_TO_SAY
User declined to state their orientation.

STRAIGHT
User identifies as straight (heterosexual).

GAY
User identifies as gay/lesbian (homosexual).

LESBIAN
An alias for *GAY*.

BISEXUAL
User identifies as bisexual.

QUESTIONING
User is questioning their orientation.

OTHER
User is some other unlisted orientation.

1.4.4 Data Classes

Some classes are just there to be data containers, this lists them.

Note: The data classes listed below are **not intended to be created by users** and are also **read-only**.

For example, this means that you should not make your own *User* instances nor should you modify the *User* instance yourself.

If you want to get one of these data classes instances they'd have to be through the client, and a common way of doing so is through the *Client.get_user()* and other function or attributes of data classes that you receive from the events specified in *Client*.

Warning: Nearly all data classes here have `__slots__` defined which means that it is impossible to have dynamic attributes to the data classes.

More information about `__slots__` can be found in the official python documentation.

IMVUObject

class imvu.**IMVUObject**

id
Get the unique ID of the object.

Returns The unique integer ID given by IMVU

User

class imvu.**User**
Represents an IMVU user.

username
Username of a user.

created
The date and time a user registered their account.

Returns Datetime that the user registered.

Return type datetime.datetime

id
Get the unique ID of the object.

Returns The unique integer ID given by IMVU

Avatar

class imvu.**Avatar**
Represents an IMVU avatar.

created
The date and time a user registered their account.

Returns Datetime that the user registered.

Return type datetime.datetime

id
Get the unique ID of the object.

Returns The unique integer ID given by IMVU

look_url
Get the look URL for an avatar.

Returns The API URL to the look.

Return type str

products
Get a list of products that the avatar is wearing.

Returns A list containing each *Product* an avatar is wearing.

Return type list

Product

class imvu.**Product**
Represents an IMVU product.

creator
The creator of a *Product*.

Returns The *User* that created the product.

Return type *User*

download(*output=None*, *full=False*)
Download a *Product*.

Parameters

- **output** (*str or None*) – The path to output the files.
- **full** (*bool or None*) – Download the entire derivation tree of a product.

id
Get the unique ID of the object.

Returns The unique integer ID given by IMVU

1.4.5 Exceptions

The following exceptions are thrown by the library.

exception imvu.**IMVUException**

exception imvu.**DownloadError**

CHAPTER 2

Extensions

2.1 `imvu.ext.outfit` – Outfit Parser and Viewer

`imvu.py` offers the ability to create more complex programs based on user and product data. The most common use case for the library will most likely be involving parsing room URLs to gather hidden user outfits and products. To ease this process, `imvu.py` provides an `outfit` extension library that handles room URL ("Products in Scene") parsing for you.

2.1.1 API Reference

The following section outlines the API of imvu.py's outfit extension module

Viewer

class imvu.ext.outfit.**Viewer**(*client: imvu.client.Client*)

parse(*url: str*, *hidden=False*) → typing.List[imvu.avatar.Avatar]
Parse a Products in Scene URL and return a list of *imvu.Avatar*.

Parameters

- **url** (*str*) – The Products in Scene URL.
- **hidden** (*bool or None*) – Attempt to parse hidden products.

Returns `list` containing each *imvu.Avatar* found in the scene.

Return type list[*imvu.Avatar*]

Index

Symbols

__version__ (in module imvu), 6

A

Availability (class in imvu), 7
AVAILABLE (imvu.Availability attribute), 7
Avatar (class in imvu), 10
AWAY (imvu.Availability attribute), 7

B

BISEXUAL (imvu.Orientation attribute), 9

C

Client (class in imvu), 6
created (imvu.Avatar attribute), 10
created (imvu.User attribute), 9
creator (imvu.Product attribute), 10

D

DISPLAY_ONLY (imvu.Visibility attribute), 8
DIVORCED (imvu.RelationshipStatus attribute), 8
DND (imvu.Availability attribute), 7
DO_NOT_DISTURB (imvu.Availability attribute), 7
download() (imvu.Product method), 10
DownloadError, 11

F

FEMALE (imvu.Gender attribute), 8

G

GAY (imvu.Orientation attribute), 8
Gender (class in imvu), 8
get_avatar() (imvu.Client method), 6
get_product() (imvu.Client method), 7
get_raw() (imvu.Client method), 7
get_user() (imvu.Client method), 7

H

HIDDEN (imvu.Gender attribute), 8
HIDDEN (imvu.Visibility attribute), 8

I

id (imvu.Avatar attribute), 10
id (imvu.IMVUObject attribute), 9
id (imvu.Product attribute), 10
id (imvu.User attribute), 10
IMVUException, 11
IMVUObject (class in imvu), 9

L

LESBIAN (imvu.Orientation attribute), 9
look_url (imvu.Avatar attribute), 10

M

MALE (imvu.Gender attribute), 8
MARRIED (imvu.RelationshipStatus attribute), 8

O

OFFLINE (imvu.Availability attribute), 8
ONLINE (imvu.Availability attribute), 7
Orientation (class in imvu), 8
OTHER (imvu.Orientation attribute), 9
OTHER (imvu.RelationshipStatus attribute), 8

P

parse() (imvu.ext.outfit.Viewer method), 13
PREFER_NOT_TO_SAY (imvu.Orientation attribute), 8
PREFER_NOT_TO_SAY (imvu.RelationshipStatus attribute), 8
Product (class in imvu), 10
products (imvu.Avatar attribute), 10

Q

QUESTIONING (imvu.Orientation attribute), 9

R

RELATIONSHIP (imvu.RelationshipStatus attribute), 8
RelationshipStatus (class in imvu), 8

REMOVED (imvu.Visibility attribute), 8

S

SEEING_SOMEONE (imvu.RelationshipStatus attribute), 8
SINGLE (imvu.RelationshipStatus attribute), 8
STRAIGHT (imvu.Orientation attribute), 8

U

User (class in imvu), 9
username (imvu.User attribute), 9

V

version_info (in module imvu), 6
Viewer (class in imvu.ext.outfit), 13
Visibility (class in imvu), 8
VISIBLE (imvu.Visibility attribute), 8