pyCraft Documentation

Release 0.7.0

Ammar Askar

Sep 12, 2020

Contents

`pyCraft` is a python project to handle networking between a Minecraft server as a client.

The authentication package contains utilities to manage communicating with Mojang's authentication servers in order to log in with a minecraft account, edit profiles etc

The Connection class under the networking package handles connecting to a server, sending packets, listening for packets etc

Contents:

CHAPTER 1

Authentication

The authentication module contains functions and classes to facilitate interfacing with Mojang's Yggdrasil authentication service.

1.1 Logging In

The most common use for this module in the context of a client will be to log in to a Minecraft account. The first step to doing this is creating an instance of the AuthenticationToken class after which you may use the authenticate method with the user's username and password in order to make the AuthenticationToken valid.

class `minecraft.authentication.`**`AuthenticationToken`**(*username=None*, *access_token=None*, *client_token=None*)

Represents an authentication token.

See http://wiki.vg/Authentication.

Constructs an *AuthenticationToken* based on *access_token* and *client_token*.

Parameters: access_token - An *str* object containing the *access_token*. client_token - An *str* object containing the *client_token*.

Returns: A *AuthenticationToken* with *access_token* and *client_token* set.

`authenticate`(*username*, *password*, *invalidate_previous=False*)

Authenticates the user against https://authserver.mojang.com using *username* and *password* parameters.

Parameters:

username - An *str* object with the username (unmigrated accounts) or email address for a Mojang account.

password - An *str* object with the password. invalidate_previous - A *bool*. When *True*, invalidate

all previously acquired 'access_token's across all clients.

Returns: Returns *True* if successful. Otherwise it will raise an exception.

Raises: minecraft.exceptions.YggdrasilError

Upon success, the function returns True, on failure a YggdrasilError is raised. This happens, for example if an incorrect username/password is provided or the web request failed.

exception `minecraft.authentication.`**`YggdrasilError`**(*message=None*, *status_code=None*, *yggdrasil_error=None*, *yggdrasil_message=None*, *yggdrasil_cause=None*)

Base *Exception* for the Yggdrasil authentication service.

Parameters

- **message** (*str*) – A human-readable string representation of the error.
- **status_code** (*int*) – Initial value of *status_code*.
- **yggdrasil_error** (*str*) – Initial value of *yggdrasil_error*.
- **yggdrasil_message** (*str*) – Initial value of *yggdrasil_message*.
- **yggdrasil_cause** (*str*) – Initial value of *yggdrasil_cause*.

status_code = None

int or *None*. The associated HTTP status code. May be set.

yggdrasil_cause = None

str or *None*. The *"cause"* field of the Yggdrasil response: a string containing additional information about the error. May be set.

yggdrasil_error = None

str or *None*. The *"error"* field of the Yggdrasil response: a short description such as *"Method Not Allowed"* or *"ForbiddenOperationException"*. May be set.

yggdrasil_message = None

str or *None*. The *"errorMessage"* field of the Yggdrasil response: a longer description such as *"Invalid credentials. Invalid username or password."*. May be set.

1.2 Arbitrary Requests

You may make any arbitrary request to the Yggdrasil service with the _make_request method passing in the AUTH_SERVER as the server parameter.

`minecraft.authentication.`**`AUTH_SERVER = 'https://authserver.mojang.com'`**

The base url for Ygdrassil requests

`minecraft.authentication.`**`_make_request`**(*server*, *endpoint*, *data*)

Fires a POST with json-packed data to the given endpoint and returns response.

Parameters: endpoint - An *str* object with the endpoint, e.g. "authenticate" data - A *dict* containing the payload data.

Returns: A *requests.Request* object.

1.2.1 Example Usage

An example of making an arbitrary request can be seen here:

```
payload = {'username': username,
           'password': password}

authentication._make_request(authentication.AUTH_SERVER, "signout", payload)
```

CHAPTER 2

Connecting to Servers

Your primary dealings when connecting to a server will be with the Connection class

2.1 Writing Packets

The packet class uses a lot of magic to work, here is how to use them. Look up the particular packet you need to deal with, for this example let's go with the `serverbound.play.KeepAlivePacket`

Pay close attention to the definition attribute, and how our class variable corresponds to the name given from the definition:

```
from minecraft.networking.packets import serverbound
packet = serverbound.play.KeepAlivePacket()
packet.keep_alive_id = random.randint(0, 5000)
connection.write_packet(packet)
```

and just like that, the packet will be written out to the server.

It is possible to implement your own custom packets by subclassing `minecraft.networking.packets.Packet`. Read the docstrings and in packets.py and follow the examples in its subpackages for more details on how to do advanced tasks like having a packet that is compatible across multiple protocol versions.

2.2 Listening for Certain Packets

Let's look at how to listen for certain packets, the relevant decorator being

A decorator can be used to register a packet listener:

Example usage:

```
connection = Connection(options.address, options.port, auth_token=auth_token)
connection.connect()

from minecraft.networking.packets.clientbound.play import ChatMessagePacket

@connection.listener(ChatMessagePacket)
def print_chat(chat_packet):
    print "Position: " + str(chat_packet.position)
    print "Data: " + chat_packet.json_data
```

Altenatively, packet listeners can also be registered seperate from the function definition.

An example of this can be found in the `start.py` headless client, it is recreated here:

```
connection = Connection(options.address, options.port, auth_token=auth_token)
connection.connect()

def print_chat(chat_packet):
    print "Position: " + str(chat_packet.position)
    print "Data: " + chat_packet.json_data

from minecraft.networking.packets.clientbound.play import ChatMessagePacket
connection.register_packet_listener(print_chat, ChatMessagePacket)
```

The field names `position` and `json_data` are inferred by again looking at the definition attribute as before

Python Module Index

Index