pyCraft Documentation

Release 0.0.1

Ammar Askar

April 02, 2015

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 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 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 convenience method

should be used which will return a LoginResponse object. See LoginResponse for more details on the returned attributes

or raise a YggdrasilError on failure, for example if an incorrect username/password is provided or the web request failed

1.2 Arbitary Requests

You may make any arbitary request to the Yggdrasil service with

1.2.1 Example Usage

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

```
url = authentication.BASE_URL + "session/minecraft/join"
server_id = encryption.generate_verification_hash(packet.server_id, secret, packet.public_key)
payload = {'accessToken': self.connection.login_response.access_token,
           'selectedProfile': self.connection.login_response.profile_id,
           'serverId': server_id}

authentication.make_request(url, payload)
```

CHAPTER 2

Connecting to Servers

Your primary dealings when connecting to a server will deal 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 my example let's go with the `KeepAlivePacket`

Pay close attention to the definition attribute, we're gonna be using that to assign values within the packet:

```
packet = 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

2.2 Listening for Certain Packets

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

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

```
connection = Connection(address, port, login_response)
connection.connect()

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

from network.packets 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

n

N