Mumpy Documentation

Ian Ling

Feb 03, 2020

Contents

CHAPTER 1

User Guide

1.1 Installation

Mumpy has not been released on PyPI yet, so it must be cloned from the Git repo and installed manually.

```
$ pip3 install --user git+https://github.com/ianling/mumpy.git
```

1.1.1 Requirements

- **Python 3.6+**
  - opuslib
  - pycryptodome
  - protobuf
- **libopus (for audio)**
  - Debian/Ubuntu: *apt install libopus0*
  - OSX: *brew install opus*
  - Windows: http://opus-codec.org/downloads/

1.2 Usage

Mumpy is both a fully-featured Mumble client and a server. As a client, it offers both an API for interacting with a Mumble server and an event-driven framework for reacting to various actions and situations as they occur.

This same API is offered from the server's perspective as well, and is also event-driven. This gives you full control over how the server behaves in any situation, and allows you to easily expand functionality beyond what is offered by the official Mumble server.

The API contains all the features you would expect from Mumble, such as the ability to send and receive voice and text chat messages, kick/ban/mute/deafen users, create/edit/remove channels, and most everything else you can do in the official Mumble client and server.

```
import mumpy

client = mumpy.Client()
client.connect('localhost', port=64738)
client.text_message("I am sending a text chat message to my current channel.")
client.kick_user("BadUser1337", reason="Not good.")

# you can also interact with User and Channel objects in intuitive ways
bad_user = client.get_user('BadUser1337')
bad_user.kick(ban=True)

my_current_channel = client.channel
my_current_channel.rename('New Channel Name')
```

A full list of all the methods available can be found in the *API Reference* section of the documentation.

The event-driven portion is essentially an alert system that allows you to run your own code in response to specific events happening. Some of these events include users connecting or disconnecting, people sending voice or text chat messages, people being kicked or banned, and new channels being created or removed.

A full list of all the events you can add handlers for can be found in the *`EventType`* part of the *API Reference* section.

Event handlers should always accept one parameter, an `Event` object. This object will contain a number of attributes that provide more information about the event that occurred, such as:

- `.type` - the type of event, from the *`EventType`* enum
- `.server` - the `Client` instance that the event originated from
- **`.raw_message` - the raw protobuf message object that caused the event to fire.** The fields you can expect to see in each protobuf message type are documented in the official Mumble client's protobuf definition file

Example event handler for the *`USER_KICKED`* event.

```
def kick_event_handler(event):
    kicker = event.server.get_user(raw_message.actor)
    victim = event.server.get_user(raw_message.session)
    print(f"{kicker.name} kicked {victim.name} from the server!")

my_client.add_event_handler(EventType.USER_KICKED, kick_event_handler)
```

Many parts of Mumpy operate asynchronously, so some of the functions do not return values themselves. For example, when you call the `Client.update_user_stats()` method, a request for the user's stats is sent to the server. The server will eventually (usually within milliseconds) respond, which will trigger the *`USER_STATS_UPDATED`* event, where you can handle the values that the server sent back to us.

A (non-exhaustive) list of events that each function is expected to cause is included in each function's documentation in the *API Reference* section. If you would like a log all the events Mumpy is firing in real time, enable DEBUG logging output. See the *Logging* section below for more details.

1.2.1 SSL Certificates

Mumble allows clients to use an SSL certificate to verify their identity on the server. This also allows the server to remember which channel they were last in when they disconnected, and assign them various permissions on the server.

You can generate a self-signed SSL certificate and key file using a command like the following:

```
$ openssl req -newkey rsa:2048 -nodes -keyout mumpy_key.pem -x509 -days 2000 -out␣
↪mumpy_certificate.pem
```

To use the certificate and key file you generated, use the `certfile` and `keyfile` parameters when connecting to a server:

```
import mumpy
my_client = mumpy.Client()
my_client.connect('localhost', certfile='mumpy_certificate.pem', keyfile='mumpy_key.
↪pem')
```

Likewise, the server portion of Mumpy must provide a certificate identifying itself to clients when they connect.

```
import mumpy
my_server = mumpy.Server('server_certificate.pem', 'server_key.pem')
```

1.2.2 Logging

Mumpy uses Python's logging library to handle logging. If you are seeing too many logs, you can add the following code to your program to reduce the logging verbosity:

```
import logging

logging.basicConfig(level=logging.WARNING)  # DEBUG, INFO, and ERROR are also valid
```

1.3 Client Examples

1.3.1 Barebones Connection

This example simply connects to a server, sends a text chat message to the channel, and then disconnects.

```
import mumpy

my_bot = mumpy.Client(username="MyBot")
my_bot.connect('localhost')  # port=64738 by default
my_bot.text_message("HELLO!")
my_bot.disconnect()
```

1.3.2 Barebones Connection Using 'with'

This example uses a different syntax to perform all the same actions as the example above.

```
import mumpy

with mumpy.Client() as my_bot:
    my_bot.connect('localhost')
    my_bot.text_message("Hello!")
```

1.3.3 Echo Bot

This example is a bot that echoes all text chat messages back to the original sender as a private message.

```
import mumpy
from time import sleep

def text_message_handler(event):
    sender = event.server.get_user_by_id(raw_message.actor)
    message_body = event.raw_message.message
    sender.text_message(message_body)

my_bot = mumpy.Client(username="MyBot")
my_bot.add_event_handler(mumpy.EventType.MESSAGE_RECEIVED, text_message_handler)  #␣
↪add our function as a handler for MESSAGE_RECEIVED events
my_bot.connect('localhost')

while my_bot.is_alive():
    sleep(1)
```

1.3.4 Play WAV File

This example is a bot that connects to a server, waits for the UDP socket to become established, and then immediately transmits a WAV file using the `udp_connected_handler` function. At the moment, WAV files must be in 48kHz 16-bit format.

```
import mumpy
from time import sleep

def udp_connected_handler(event):
    event.server.play_wav('/home/ian/some_sound.wav')

my_bot = mumpy.Client(username="MyBot")
my_bot.add_event_handler(mumpy.EventType.UDP_CONNECTED, udp_connected_handler)
my_bot.connect('localhost')

while my_bot.is_alive():
    sleep(1)
```

1.4 Server Examples

Coming soon...

1.5 Development

Mumpy is open source under the GNU General Public License (GPL) version 3. The source code can be found on Github.

1.5.1 Contributing

If you have any contributions to make, whether they are bug reports, feature requests, or even code, feel free to submit issues and pull requests on Github.

This repo uses Travis CI to run a Python style checker called flake8, which looks for errors in the code, as well as deviations from the PEP8 style guide.

In order to style check your code locally before pushing it to Github, you can run a command like the following, from the root of the repo:

```
$ python3 -m flake8 .
```

We also ignore some of the flake8 style suggestions. Check the Travis config file in the repo to see exactly what flake8 command will get run on code pushed to the repo.

1.5.2 Building the Documentation

To build the documentation locally, enter the `docs/` directory and run the command `make html`.

This section contains information about installing and using Mumpy.

CHAPTER 2

API Reference

2.1 Channel Object

class mumpy.channel.**Channel**(*server*, *message=None*)

get_description()
Queries the server for the channel's description.

Returns None

get_users()
Retrieves a list of Users in this channel.

Returns a list of the Users in this channel

Return type list

id
This channel's ID.

remove()
Removes this channel from the server.

Returns None

rename(*new_name*)
Sets the channel's name to new_name.

Parameters **new_name** (*str*) – The new name for the channel

Returns None

text_message(*message*)
Sends a text message to this channel.

Parameters **message** (*str*) – the message

Returns None

2.2 EventType Enum

class mumpy.constants.**EventType**

The event types supported by Mumpy.

AUDIO_DISABLED = 'audio_disabled'

Fired when the client disables audio processing. This happens when the client fails to initialize the chosen audio codec, or does not support any of the server's audio codecs.

AUDIO_ENABLED = 'audio_enabled'

Fired when the client enables audio processing. This happens when the client initially connects to the server and successfully initializes an audio codec.

AUDIO_TRANSMISSION_RECEIVED = 'audio_transmission_received'

Fired when the client has received a complete audio transmission from the server.

AUDIO_TRANSMISSION_SENT = 'audio_transmission_sent'

Fired when the client has sent a complete audio transmission to the server.

BANLIST_MODIFIED = 'banlist_modified'

Fired when the server's ban list is modified.

CHANNEL_ADDED = 'channel_added'

Fired when a channel is added to the server.

CHANNEL_PERMISSIONS_UPDATED = 'channel_permissions_updated'

Fired when the Mumpy instance's permissions in a channel have changed.

CHANNEL_REMOVED = 'channel_removed'

Fired when a channel is removed from the server.

CHANNEL_UPDATED = 'channel_updated'

Fired when a channel is updated or modified in some way.

CONNECTED = 'self_connected'

Fired when the client has connected and authenticated successfully.

DISCONNECTED = 'self_disconnected'

Fired when the client has disconnected from the server. May be preceded by a USER_KICKED and a USER_BANNED event.

MESSAGE_RECEIVED = 'message_received'

Fired when a text message is received.

MESSAGE_SENT = 'message_sent'

Fired when the client sends a text message.

REGISTERED_USER_LIST_RECEIVED = 'registered_user_list_received'

Fired when the client receives the list of registered users on the server. These are stored in <Mumpy instance>.registered_users

UDP_CONNECTED = 'udp_connected'

Fired when the client has successfully established a UDP connection to the server

UDP_DISCONNECTED = 'udp_disconnected'

Fired when the client has lost or intentionally ended the UDP connection. This implies that audio communications have reverted back to using the TCP connection.

USER_BANNED = 'user_banned'

Fired when anyone is banned from the server.

USER_COMMENT_UPDATED = 'user_comment_updated'
Fired when a user changes their comment.

USER_CONNECTED = 'user_connected'
Fired when someone else connects to the server.

USER_DEAFENED = 'user_deafened'
Fired when a user is deafened server side (e.g. by a server admin).

USER_DISCONNECTED = 'user_disconnected'
Fired when someone else disconnects from the server. May be preceded by a USER_KICKED and a USER_BANNED event.

USER_KICKED = 'user_kicked'
Fired when anyone is kicked from the server.

USER_MUTED = 'user_muted'
Fired when a user is muted server side (e.g. by a server admin).

USER_RECORDING = 'user_recording'
Fired when a user starts recording.

USER_REGISTERED = 'user_registered'
Fired when a user registers on the server.

USER_SELF_DEAFENED = 'user_self_deafened'
Fired when a user deafens themselves.

USER_SELF_MUTED = 'user_self_muted'
Fired when a user mutes themselves.

USER_SELF_UNDEAFENED = 'user_self_undeafened'
Fired when a user undeafens themselves.

USER_SELF_UNMUTED = 'user_self_unmuted'
Fired when a user unmutes themselves.

USER_STATS_UPDATED = 'user_stats_updated'
Fired when updated stats about a user are received. This happens after the client specifically requests stats about a user.

USER_STOPPED_RECORDING = 'user_stopped_recording'
Fired when a user stops recording.

USER_TEXTURE_UPDATED = 'user_texture_updated'
Fired when a user changes their texture (avatar).

USER_UNDEAFENED = 'user_undeafened'
Fired when a user is undeafened server side (e.g. by a server admin).

USER_UNMUTED = 'user_unmuted'
Fired when a user is unmuted server side (e.g. by a server admin).

USER_UNREGISTERED = 'user_unregistered'
Fired when a user is unregistered on the server.

2.3 Mumpy Object

2.4 User Object

class mumpy.user.**User**(*server*, *message*)

channel
This user's current channel.

Returns the user's current channel

Return type *Channel*

clear_audio_log()
Clears this user's audio log, removing all their completed audio transmissions from memory.

Returns None

deafen()
Deafens the user.

Returns None

kick(*reason=''*, *ban=False*)
Kicks user. Bans the user if ban is True.

Parameters

- **reason** (*str*) – The reason for kicking
- **ban** (*bool*) – Whether or not the user should also be banned

Returns None

move_to_channel(*channel*)
Moves the user to the specified channel.

Parameters **channel** (Channel) – the Channel to move them to

Returns None

mute()
Mutes the user.

Returns None

register()
Registers the user on the server.

Returns None

session_id
This user's session ID.

Returns session ID

Return type int

text_message(*message*)
Sends the user a private text message.

Parameters **message** (*str*) – the message

Returns None

undeafen()
Undeafens the user.

Returns None

unmute()
Unmutes the user.

Returns None

unregister()
Unregisters the user on the server.

Returns None

update_comment()
Query the server for this user's comment.

Returns None

update_stats()
Requests updated stats about the user from the server.

Returns None

update_texture()
Query the server for this user's texture.

Returns None

This section contains information about the functions that Mumpy exposes to developers.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index