discodo

Release v3.0.0

kijk2869

Apr 26, 2021

INTRODUCTION

1 Features 3

2 Supported sources 5

3 Client libraries 7

Python Module Index 49

HTTP Routing Table 51

Index 53

Discodo is an enhanced audio player for discord.

CHAPTER

ONE

FEATURES

- Standalone Audio Node
- Youtube Related Video Autoplay
- Crossfade and Audio effects
- Synced Youtube Video Subtitle

CHAPTER

TWO

SUPPORTED SOURCES

- All sources that can be extracted from youtube-dl
- All formats that can be demuxed by libav

CHAPTER

THREE

CLIENT LIBRARIES

- discodo (Python)
- discodo.js (Under Developing) (Node.js)

3.1 Installation

3.1.1 Prerequisites

Discodo works with **Python 3.7 or higher.**

Ealier versions of Python may not be worked.

3.1.2 Dependencies

On Linux environments, below dependencies are required:

- `python3-dev`
- `libopus-dev`
- `libnacl-dev`

PyAV depends upon several libraries from FFmpeg:

- `libavcodec-dev`
- `libavdevice-dev`
- `libavfilter-dev`
- `libavformat-dev`
- `libavutil-dev`
- `libswresample-dev`
- `libswscale-dev`
- `pkg-config`

Mac OS X

```
$ brew install opus pkg-config ffmpeg
```

Ubuntu

```
$ sudo apt update

# Our general dependencies
$ sudo apt install --no-install-recommends -y python3-dev libopus-dev libnacl-dev

# PyAV dependencies
$ sudo apt install --no-install-recommends -y \
    pkg-config libavformat-dev libavcodec-dev libavdevice-dev \
    libavutil-dev libswscale-dev libavresample-dev libavfilter-dev
```

3.1.3 Installing

PyPI

```
$ python3 -m pip install -U discodo
```

Docker

```
$ docker pull kijk2869/discodo:release-2.3.13
```

3.1.4 Execution

You can see additional options with the `--help` flag.

```
$ python3 -m discodo [-h] [--version] [--config CONFIG] [--config-json CONFIG_JSON] [-
↪-host HOST] [--port PORT]
           [--auth AUTH] [--ws-interval WS_INTERVAL] [--ws-timeout WS_TIMEOUT] [--ip␣
↪IP] [--exclude-ip EXCLUDE_IP]
           [--default-volume DEFAULT_VOLUME] [--default-crossfade DEFAULT_CROSSFADE]
           [--default-autoplay DEFAULT_AUTOPLAY] [--bufferlimit BUFFERLIMIT] [--
↪preload PRELOAD]
           [--timeout TIMEOUT] [--enabled-resolver ENABLED_RESOLVER] [--spotify-id␣
↪SPOTIFY_ID]
           [--spotify-secret SPOTIFY_SECRET] [--verbose]
```

3.1.5 Options

```
optional arguments:
-h, --help            show this help message and exit
--version             Config json file path (default: None)
--config CONFIG       Config json file path (default: None)
--config-json CONFIG_JSON
                        Config json string (default: None)

Webserver Option:
--host HOST, -H HOST  the hostname to listen on (default: 0.0.0.0)
--port PORT, -P PORT  the port of the webserver (default: 8000)
--auth AUTH, -A AUTH  the password of the webserver (default: hellodiscodo)
--ws-interval WS_INTERVAL
                        heartbeat interval between discodo server and client␣
↪(default: 15)
--ws-timeout WS_TIMEOUT
                        seconds to close connection there is no respond from client␣
↪(default: 60)

Network Option:
--ip IP               Client-side IP blocks to use
--exclude-ip EXCLUDE_IP
                        Client-side IP addresses not to use

Player Option:
--default-volume DEFAULT_VOLUME
                        player's default volume (default: 100)
--default-crossfade DEFAULT_CROSSFADE
                        player's default crossfade seconds (default: 10.0)
--default-autoplay DEFAULT_AUTOPLAY
                        player's default auto related play state (default: True)
--bufferlimit BUFFERLIMIT
                        seconds of audio will be loaded in buffer (default: 5)
--preload PRELOAD     seconds to load next song before this song ends (default: 10)
--timeout TIMEOUT     seconds to cleanup player when connection of discord terminated␣
↪(default: 300)

Extra Extractor Option:
--enabled-resolver ENABLED_RESOLVER
                        Extra resolvers to enable (Support melon and spotify)
--spotify-id SPOTIFY_ID
                        Spotify API id (default: None)
--spotify-secret SPOTIFY_SECRET
                        Spotify API secret (default: None)

Logging Option:
--verbose, -v         Print various debugging information
```

Config file

```
{
    "HOST": "0.0.0.0",
    "PORT": 8000,
    "PASSWORD": "hellodiscodo",
    "HANDSHAKE_INTERVAL": 15,
    "HANDSHAKE_TIMEOUT": 60,
    "IPBLOCKS": [],
    "EXCLUDEIPS": [],
    "DEFAULT_AUTOPLAY": true,
    "DEFAULT_VOLUME": 1,
    "DEFAULT_CROSSFADE": 10,
    "DEFAULT_GAPLESS": false,
    "BUFFERLIMIT": 5,
    "PRELOAD_TIME": 10,
    "VCTIMEOUT": 300,
    "ENABLED_EXT_RESOLVER": [
        "melon",
        "vibe"
    ],
    "SPOTIFY_ID": null,
    "SPOTIFY_SECRET": null
}
```

3.2 Quickstart

3.2.1 A Minimal Bot with discord.py

Let's make a bot which uses local node feature.

It looks like this:

```
import discord
import discodo

client = discord.Client()
codo = discodo.DPYClient(client)

@client.event
async def on_ready():
    print(f"I logged in as {client.user} (client.user.id)")

@codo.event("SOURCE_START")
async def sendPlaying(VC, Data):
    await VC.channel.send(f"I'm now playing {Data['source']['title']}")

@client.event
async def on_message(message):
    if message.author == client.user:
        return

    if message.content.startswith("!join"):
        if not message.author.voice:
            return await message.channel.send("Join the voice channel first.")
```

(continues on next page)

(continued from previous page)

```
        await codo.connect(message.author.voice.channel)

        return await message.channel.send(f"I connected to {message.author.voice.
↪channel.mention}")

    if message.content.startswith("!play"):
        VC = codo.getVC(message.guild, safe=True)

        if not VC:
            return await message.channel.send("Please type `!join` first.")

        if not hasattr(VC, "channel"):
            VC.channel = message.channel

        source = await VC.loadSource(message.content[5:].strip())

        if isinstance(source, list):
            return await message.channel.send(f"{len(source) - 1} songs except
↪{source[0].title} added.")
        else:
            return await message.channel.send(f"{source.title} added.")

    if message.content.startswith("!stop"):
        VC = codo.getVC(message.guild, safe=True)

        if not VC:
            return await message.channel.send("I'm not connected to any voice channel␣
↪now.")

        await VC.destroy()

        return await message.channel.send("I stopped the player and cleaned the queue.
↪")

codo.registerNode()
client.run("your discord bot token here")
```

Let's name this file `simple_musicbot.py`

Assume you know how to use discord.py, and I will explain the discodo code step by step.

1. We create an instance of `DPYClient`. This client will manage voice connections to Discord.
2. After `on_ready` event, we use the `DPYClient.event()` decorator to register an event like discord.py . In this case, `SOURCE_START` will be called when the music starts playing.
3. When the `!join` command is excuted, we check if the `discord.Message.author` is connected to the voice channel. If it is, then we connected to the channel using `DPYClient.connect()`
4. When the `!play` command runs, set the VC.channel to the current message channel to send messages during playback, search for queries and add them to the list.
5. If the `!stop` command is excuted, we destroy the voice client via `VoiceClient.destroy()`
6. Finally, we set local nodes to be used by not giving host argument to `DPYClient.registerNode()`

Now that we've made a simple music bot, we have to run this. Just as you do when you run a discord.py Bot

```
$ python simple_musicbot.py
```

Now you can try playing around with your basic musicbot.

3.3 Setting Up Logging

3.3.1 Basic logging

discodo logs several information via the logging python module like discord.py. It is strongly recommended to configure the logging module, as you can't see some error if it is not set up. You can configure the logging module as simple as:

```
import logging

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

Placed at the start of the code. This will output the logs from all libraries which use the logging module, including discodo, to the console.

The `level` argument specifies what level of events to log and can be any of `CRITICAL`, `ERROR`, `WARNING`, `INFO`, and `DEBUG` and default value is `WARNING`

For more information, check the documentation of the logging module.

3.3.2 Use rich formatter

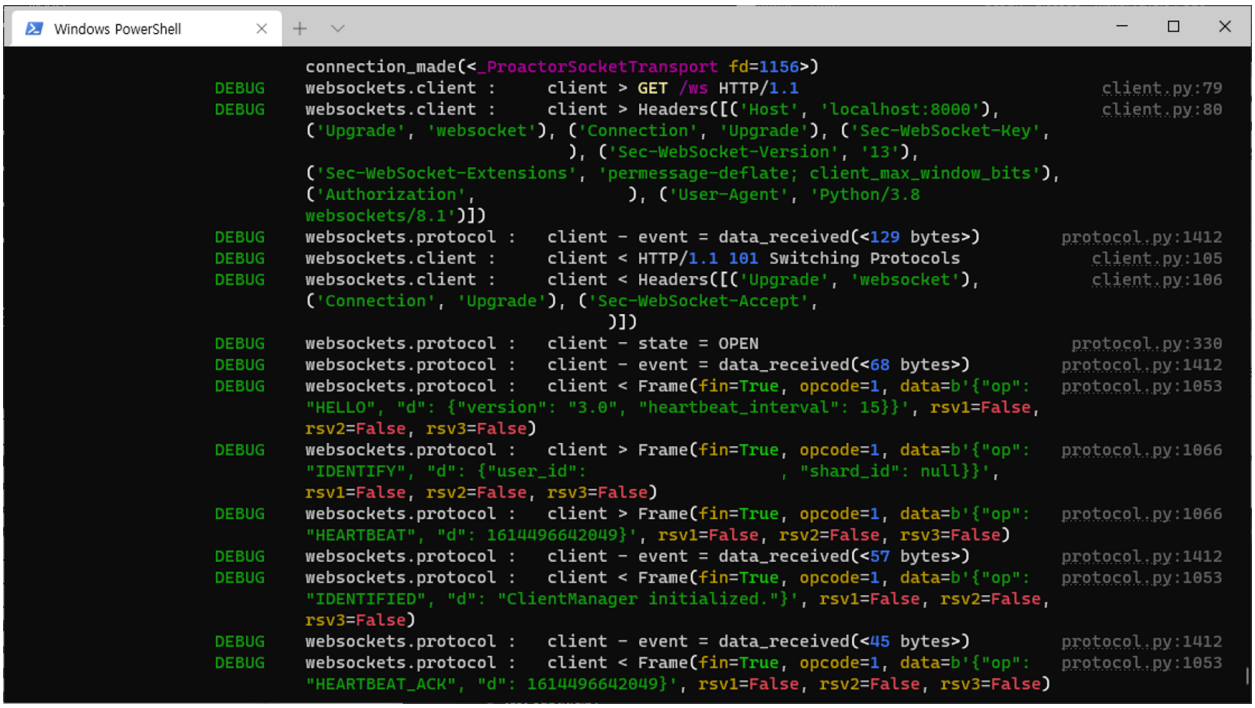
Also, discodo uses rich to improve log readability. rich is an library for rich text and formatting in the terminal. To output the log to the terminal using rich, you can set it as follows:

```
import logging
from rich.logging import RichHandler

logging.basicConfig(level=logging.INFO, format="%(name)s :\t%(message)s",␣
↪handlers=[RichHandler(rich_tracebacks=True)])
```

Because rich displays the `log level` separately, remove the `level` from the `format` argument, and set `rich_tracebacks` to `True` for formatting tracebacks.

When you set this, the log will be formatted as follows:



3.4 Websocket Connection

Discodo using websocket to send and receive events.

3.4.1 Payloads

Payload Structure

Field	Type	Description
op	string	operation name for the payload
d	?mixed (Mostly JSON)	event data

3.4.2 Connecting to Discodo

Connecting

Websocket Headers

Field	Type	Description
Authorization	string	Password for discodo server

Once connected, the client should immediately receive `HELLO` with the connection's heartbeat interval unless you missed headers or mismatched, otherwise receive `FORBIDDEN`.

> Example HELLO

```
{
    "op": "HELLO",
    "d": {
        "heartbeat_interval": 15.0
    }
}
```

> Example FORBIDDEN

```
{
    "op": "FORBIDDEN",
    "d": "why the connection forbidden"
}
```

Heartbeating

Recieveing `HELLO` payload, the client should begin sending `HEARTBEAT` every `heartbeat_interval` seconds, until the connection closed.

< Example HEARTBEAT

```
{
    "op": "HEARTBEAT",
    "d": 0 // timestamp
}
```

Event Data (`d`) can be `None`, the server will echo them.

> Example HEARTBEAT_ACK

```
{
    "op": "HEARTBEAT_ACK",
    "d": 0 // timestamp
}
```

Identifying

The client must send `IDENTIFY` to configure the audio manager before using.

< Example IDENTIFY

```
{
    "op": "IDENTIFY",
    "d": {
        "user_id": "my bot id"
        "shard_id": null // shard id
    }
}
```

Resumed

If the same `user_id` with the same `shard_id` is connected before `VC_TIMEOUT`, it will be resumed.

> Example Resumed

```
{
    "op": "RESUMED",
    "d": {
        "voice_clients": [
            [0, 0] // guild_id, voice_channel_id(can be null)
        ]
    }
}
```

When the client recieve `RESUMED`, must reconnect to each voice channel.

Disconnecting

If the connection is closed, the server will clean up manager and sources after `VC_TIMEOUT`

3.5 Voice Client

3.5.1 Get State

> Example getState

```
{
    "op": "getState",
    "d": {
        "guild_id": "guild_id"
    }
}
```

< Example getState response

```
{
    "op": "getState",
    "d": {
        "id": "VoiceClient ID",
        "guild_id": "Guild ID",
        "channel_id": "Voice Channel ID",
        "state": "Current Player State",
        "current": "Current AudioSource Object",
        "duration": "Current source duration",
        "position": "Current source position",
        "remain": "Current source remain",
        "remainQueue": "Queue length",
        "options": {
            "autoplay": "autoplay boolean",
            "volume": "volume float",
            "crossfade": "crossfade float",
            "filter": {}
        },
        "context": {}
    }
}
```

3.5.2 Get Context

GET /context

Get context of the voice client

Example response:

```
{
    // context
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.3 Set Context

POST /context
Set context of the voice client

Example response:

```
{
    // context
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

JSON Parameters

- **context** (*json*) – context to set

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.4 Put Source

POST /putSource
Put the source object on Queue

Example response:

```
{
    "source": {
        // source object
    }
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

JSON Parameters

- **source** (*json*) – the source object to put

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.5 Load Source

`POST /loadSource`

Search query and put it on Queue

Example response:

```
{
    "source": {
        // source object
    }
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

JSON Parameters

- **`query`** (*`string`*) – query to search

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.6 Get Options

`GET /options`

Get options of the voice_client

Example response:

```
{
    "autoplay": True,
    "volume": 1.0,
    "crossfade": 10.0,
    "filter": {}
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.7 Set Options

`POST /options`

Set options of the voice_client

Example response:

```
{
    "autoplay": True,
    "volume": 1.0,
    "crossfade": 10.0,
    "filter": {}
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

JSON Parameters

- **`?volume`** (*`float`*) – volume value
- **`?crossafde`** (*`float`*) – crossfade value
- **`?autoplay`** (*`boolean`*) – autoplay value
- **`?filter`** (*`json`*) – filter value

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.8 Get Position

GET /seek
Get position of the voice_client

Example response:

```
{
    "duration": 300.0,
    "position": 200.0,
    "remain": 100.0
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.9 Set Position (Seek)

POST /seek
Set position of the voice_client

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

JSON Parameters

- **offset** (*float*) – position to seek

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.10 Skip Current

POST /skip
Skip current of the voice_client

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.11 Pause

POST /pause
Pause current of the voice_client

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.12 Resume

POST /resume
Resume current of the voice_client

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue

- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.13 Shuffle Queue

POST /shuffle

Shuffle the queue of the voice_client

Example response:

```
{
    "entries": [
        // source object
    ]
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.5.14 Get Queue

GET /queue

Get the queue of the voice_client

Example response:

```
{
    "entries": [
        // source object
    ]
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id

- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.6 AudioData

3.6.1 AudioData Object

AudioData Structure

Field	Type	Description
_type	string	fixed value `AudioData`
tag	string	object tag (uuid)
id	string	source id
title	?string	source title
webpage_url	?string	source webpage url
thumbnail	?string	source thumbnail
url	?string	source stream url
duration	?integer	source duration
is_live	boolean	source live state
uploader	?string	source uploader
description	?string	source description
subtitles	json	source subtitles (Mostly SRV1)
chapters	json	source chapters (Only in youtube)
related	boolean	related playing state
context	json	object context
start_position	float	source start position

3.6.2 AudioSource Object

AudioSource Structure

Field	Type	Description
_type	string	fixed value `AudioSource`
tag	string	object tag (uuid)
id	string	source id
title	?string	source title
webpage_url	?string	source webpage url
thumbnail	?string	source thumbnail
url	?string	source stream url
duration	integer	source duration
is_live	boolean	source live state
uploader	?string	source uploader
description	?string	source description
subtitles	json	source subtitles (Mostly SRV1)
chapters	json	source chapters (Only in youtube)
related	boolean	related playing state
context	json	object context
start_position	float	start position
seekable	boolean	seekable state
position	?float	source current position

3.6.3 Get Current

`GET /current`

The source object that is currently playing

Example response:

```
{
    // source object
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.6.4 Get From The Queue

GET /queue/{tag_or_index}
The source object in queue

Example response:

```
{
    // source object
}
```

Parameters

- **tag_or_index** (*int or str*) – the tag or index of source object

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.6.5 Edit Current

POST /current
Edit the source object that is currently playing

Example response:

```
{
    // source object
}
```

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

JSON Parameters

- **?context** (*json*) – context to save on the object

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.6.6 Edit From The Queue

POST /queue/{tag_or_index}

Edit the source object in queue

Example response:

```
{
    // edited source object
}
```

Parameters

- **tag_or_index** (*int or str*) – the tag or index of source object

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

JSON Parameters

- **?index** (*integer*) – index to move the source in queue
- **?context** (*json*) – context to save on the object
- **?start_position** (*float*) – position to start on (only in AudioData)

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.6.7 Remove From The Queue

DELETE /queue/{tag_or_index}

Remove the source object in queue

Example response:

```
{
    "removed": {
        // removed source object
    },
    "entries": [
```

(continues on next page)

(continued from previous page)

```
        // list of source in queue
    ]
}
```

Parameters

- **tag_or_index** (*int or str*) – the tag or index of source object

Request Headers

- Authorization – Password for discodo server
- *User-ID* – the bot user id
- *?Shard-ID* – the bot shard id
- *Guild-ID* – the guild id of queue
- *VoiceClient-ID* – the voiceclient id

Status Codes

- 200 OK – no error
- 403 Forbidden – authorization failed or VoiceClient-ID mismatched
- 404 Not Found – ClientManager or VoiceClient not found

3.7 Event Reference

This section outlines the different types of events dispatched by discodo node with websocket.

Note: If you are using `DPYClient`, the events that you get will have something different. See this *Event Reference*.

3.7.1 STATUS

Called when the client requests system information by `getStatus`. the unit is mega bytes or percent.

Field	Type	Description
UsedMemory	integer	The process memory usage
TotalMemory	integer	The system memory usage
ProcessLoad	integer	The process cpu usage
TotalLoad	integer	The system cpu usage
Cores	integer	The cpu core count
Threads	integer	The process thread count
NetworkInbound	integer	The network inbound counters
NetworkOutbound	integer	The network outbound counters

3.7.2 HEARTBEAT_ACK

Called when the client send `HEARTBEAT` payload. The data of this event is payload data.

3.7.3 IDENTIFIED

Called when the new voice client has successfully created. This is not the same as the client being fully connected.

Field	Type	Description
guild_id	int	The guild id of the voice client
id	str	The id of the voice client

3.7.4 VC_DESTROYED

Called when the voice client has successfully destroyed.

Note: This does not mean that the bot have disconnected from the voice channel. When the client receives this event, it should disconnect from the voice channel.

Field	Type	Description
guild_id	int	The guild id of the voice client that is destroyed

3.7.5 QUEUE_EVENT

Called when there is something changed in the queue of the voice client.

Field	Type	Description
guild_id	int	The guild id of the voice client
name	str	The name of the method
args	list	The arguments of the method

3.7.6 VC_CHANNEL_EDITED

Called when the voice channel of the voice client is changed.

Field	Type	Description
guild_id	int	The guild id of the voice client
channel_id	int	The channel id of the voice client

3.7.7 putSource

Called when some sources are put in the queue.

Field	Type	Description
guild_id	int	The guild id of the voice client
sources	list	The sources which is put

3.7.8 loadSource

Called when some sources are searched and put in the queue.

Field	Type	Description
guild_id	int	The guild id of the voice client
source	list or AudioData	The sources which is searched and put

3.7.9 REQUIRE_NEXT_SOURCE

Called when the player needs next source to play. If you set `autoplay` as `True`, the related source will be put after this event.

Field	Type	Description
guild_id	int	The guild id of the voice client
current	AudioSource	The source which the player is currently playing

3.7.10 SOURCE_START

Called when the player starts to play the source.

Field	Type	Description
guild_id	int	The guild id of the voice client
source	AudioSource	The source which the player starts to play

3.7.11 SOURCE_STOP

Called when the player stops to play the source.

Field	Type	Description
guild_id	int	The guild id of the voice client
source	AudioSource	The source which the player stops to play

3.7.12 getState

Called when the client requests the player state by `getState`.

Field	Type	Description
guild_id	str	The guild id of the voice client
channel_id	str	The channel id of the voice client
state	str	Current state of the voice client
current	AudioSource	Current source of the player
duration	float	Current duration of the source that is playing
position	float	Current position of the source that is playing
remain	float	(duration value) - (position value)
remainQueue	int	Current queue length of the player
options	JSON	Current options of the player

3.7.13 getQueue

Called when the client requests the player queue by `getQueue`.

Field	Type	Description
guild_id	int	The guild id of the voice client
entries	list	The entries of the queue

3.7.14 requestSubtitle

Called when the client requests synced subtitles by `requestSubtitle`.

Field	Type	Description
guild_id	int	The guild id of the voice client
identify	str	The id to identify the subtitle
url	str	The url of the subtitle

3.7.15 Subtitle

This event is for sending the sync subtitle. This event is sent according to the player's position.

Field	Type	Description
guild_id	int	The guild id of the voice client
identify	str	The id to identify the subtitle
previous	str	The content of previous subtitle
current	str	The content of current subtitle
next	str	The content of next subtitle

3.7.16 subtitleDone

Called when the subtitle is done.

Field	Type	Description
guild_id	int	The guild id of the voice client
identify	str	The id to identify the subtitle

3.7.17 PLAYER_TRACEBACK

Called when the player gets some traceback while trying to send packets to discord server.

Field	Type	Description
guild_id	int	The guild id of the voice client
traceback	str	The traceback information which the player gets

3.7.18 SOURCE_TRACEBACK

Called when the player gets some traceback while trying to play the source. That source will be automatically removed from the queue.

Field	Type	Description
guild_id	int	The guild id of the voice client
source	Union[AudioData, AudioSource]	The source which the player gets traceback while trying to play
traceback	str	The traceback information which the player gets

3.8 High Level API

This section outlines the API of discodo.

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, logs will not be output anywhere. See *Setting Up Logging* for more information.

3.8.1 Client

class discodo.**DPYClient**(*client*)

Represents a client connection that connects to Discodo. This class will interact with Discodo nodes.

Parameters **class** (*discord.Client*) – The client of the bot with discord.py

Variables

- ***Nodes*** (*list*) – The list of *discodo.Node* that is registered.
- **dispatcher** (EventDispatcher) – The event dispatcher that the client dispatches events.
- **loop** (*asyncio.AbstractEventLoop*) – The event loop that the client uses for operation.

async connect(*channel: discord.channel.VoiceChannel*, *node: Optional[discodo.client.DPYClient.NodeClient] = None*) → None
Connect to the voice channel.

Parameters

- **channel** (*discord.VoiceChannel*) – The channel to connect to.
- **node** (*Optional[discodo.Node]*) – The node to connect with, defaults to `getBestNode()`

Raises

- **ValueError** – The `channel` value has no `guild` property.
- ***discodo.NodeNotConnected*** – There is no discodo node that is connected.
- **asyncio.TimeoutError** – The connection is not established in 10 seconds.

Return type *discodo.VoiceClient*

async destroy(*guild: discord.guild.Guild*) → None
Destroy the voice client and disconnect from the voice channel

Parameters **guild** (*discord.Guild*) – The guild to destroy the voice client.

Raises ***discodo.VoiceClientNotFound*** – The voice client was not found.

async disconnect(*guild: discord.guild.Guild*) → None
Disconnect from the voice channel.

Note: This coroutine doesn't destroy the voice client. Recommand to use *destroy()*

Parameters **guild** (*discord.Guild*) – The guild to disconnect from.

property event
A decorator that registers an event to listen to.

Parameters **event** (*str*) – The event name to listen to.

getBestNode(*exceptNode=None*)
Get the node with the fewest connected players.

Parameters **exceptNode** (*Optional[discodo.Node]*) – The host to except from the list.

Return type *discodo.Node*

getVC(*guild*, *safe=False*)
Get a voice client from the guild.

Parameters

- **guild** (*int or discord.Guild*) – Guild or guild ID from which to get the voice client.
- **safe** (*bool*) – Whether to raise an exception when the voice client cannot be found, defaults to `False`.

Raises ***discodo.VoiceClientNotFound*** – The voice client was not found and the `safe` value is `False`.

Return type *discodo.VoiceClient*

getWebsocket(*id*)
Get a websocket object of the shard from discord.py

Parameters **id** (*int*) – The shard id to get a object.

Return type discord.gateway.DiscordWebSocket

registerNode(*host=None*, *port=None*, *password='hellodiscodo'*, *region=None*, *launchOptions=None*)
Creates a websocket connection of the node and register it on the client.

If the value `host` or `port` is `None`, it will launch local node process.

Parameters

- **host** (*Optional[str]*) – The host of the node to connect to.
- **port** (*Optional[int]*) – The port of the node to connect to.
- **password** (*Optional[str]*) – The password of the node to connect to, defaults to `hellodiscodo`.
- **region** (*Optional[str]*) – The region of the node to connect to. This is like a annotation. It is not involved in the actual connection.
- **launchOptions** (*Optional[dict]*) – The options to use when launching a local node

Returns The scheduled task to connect with the node

Return type asyncio.Task

property voiceClients
A property that returns all voice clients from all of connected nodes.

Return type dict

3.8.2 Utils

class discodo.utils.**EventDispatcher**(*loop: Optional[asyncio.events.AbstractEventLoop] = None*)
Represents an event dispatcher similar to EventEmitter

Variables **loop** (*Optional[asyncio.AbstractEventLoop]*) – The event loop that the dispatcher uses for operation, defaults to `asyncio.get_event_loop()`

dispatch(*event_: str*, **args*, ***kwargs*) → None
Call the listeners which is matched with event name.

Parameters

- **event** (*str*) – The event name to dispatch.
- ***args** – An argument list of data to dispatch with.
- ***kwargs** – A keyword argument list of data to dispatch with.

event(*event: str*)
A decorator that registers an event to listen to.

Parameters **event** (*str*) – The event name to listen to.

off(*event: str*, *func: Callable*)
Remove the `func` function from the listeners list of the `event`

Parameters

- **event** (*str*) – The event name to remove from.
- **func** (*Callable*) – The function to remove from the list.

Return type discodo.EventDispatcher

offAny(*func: Callable*)
Remove the `func` function from the listeners list

Parameters **func** (*Callable*) – The function to remove from the list.

Return type discodo.EventDispatcher

on(*event: str*, *func: Callable*)
Adds the `func` function to the end of the listeners list of the `event`

Parameters

- **event** (*str*) – The event name to listen to.
- **func** (*Callable*) – The function to call when event dispatching.

Return type discodo.EventDispatcher

onAny(*func: Callable*)
Adds the `func` function to the end of the listeners list

Parameters **func** (*Callable*) – The function to call when event dispatching.

Return type discodo.EventDispatcher

async wait_for(*event: str*, *condition: Optional[Callable] = None*, *timeout: Optional[float] = None*)
Waits for an event that is matching with `condition` to be dispatched for `timeout`

Parameters

- **event** (*str*) – The event name to wait for
- **condition** (*Optional[Callable]*) – A predicate to check what to wait for. this function must return `bool`
- **timeout** (*Optional[float]*) – The number of seconds to wait.

Raises **asyncio.TimeoutError** – The timeout is provided and it was reached.

Return type Any

discodo.utils.status.**getCpuCount**()
Get cpu core count.

Return type int

discodo.utils.status.**getMemory**()
Get system memory usage. the unit is mega bytes.

Return type int

discodo.utils.status.**getNetworkInbound**()
Get network inbound counters. the unit is mega bytes.

Return type int

discodo.utils.status.**getNetworkOutbound**()
Get network outbound counters. the unit is mega bytes.

Return type int

discodo.utils.status.**getProcessCpu**()
Get process cpu usage. the unit is percent.

Return type int

discodo.utils.status.**getProcessMemory**()
Get process memory usage. the unit is mega bytes.

Return type int

discodo.utils.status.**getProcessThreads**()
Get process thread count.

Return type int

discodo.utils.status.**getStatus**()
Represents all information to dictionary

Return type dict

discodo.utils.status.**getTotalCpu**()
Get system cpu usage. the unit is percent.

Return type int

class discodo.utils.**CallbackList**(*iterable=()*, */*)
A list that has callback when the value is changed.

static callback(*name*, **args*)
Override this function to use callback

tcp.**getFreePort**() → int
Get free port from the system.

Return type int

3.8.3 Node

class discodo.**Node**(*client*, *host*, *port*, *user_id*, *shard_id=None*, *password='hellodiscodo'*, *region=None*)
Represents a discodo node connection.

Variables

- **client** (discodo.DPYClient) – The client which the node is binded.
- **ws** (*Optional*) – The websocket gateway the client is currently connected to.
- **dispatcher** (EventDispatcher) – The event dispatcher that the client dispatches events.
- **loop** (*asyncio.AbstractEventLoop*) – The event loop that the client uses for operation.
- **host** (*str*) – The host of the node to connect to.
- **port** (*int*) – The port of the node to connect to.
- **password** (*str*) – The password of the node to connect to.
- **user_id** (*int*) – This bot's ID
- **shard_id** (*Optional[int]*) – This bot's shard ID, could be None
- **region** (*Optional[str]*) – Region set when registering a node

- **voiceClients** (*dict*) – A dictionary consisting of pairs of guild IDs and voice clients.

property URL
Represents the restful api url of the node.

Return type str

property WS_URL
Represents the websocket url of the node.

Return type str

async close()
some action to do after disconnected from node

async connect()
Connect to the node.

Raises **ValueError** – The node is already connected.

async destroy()
Destroy the node and remove from the client.

async discordDispatch(*payload*)
Dispatch the discord payload to the node.

Note: If you are using *discodo.DPYClient*, you don't have to use this.

Parameters **payload** (*dict*) – The event data from the discord.

async getStatus()
Get status like cpu usage from the node with websocket.

Return type dict

getVC(*guildID*, *safe=False*)
Get a voice client from the guild.

Parameters

- **guildID** (*int*) – guild ID from which to get the voice client.
- **safe** (*bool*) – Whether to raise an exception when the voice client cannot be found, defaults to `False`.

Raises ***discodo.VoiceClientNotFound*** – The voice client was not found and the `safe` value is `False`.

Return type *discodo.VoiceClient*

property is_connected
Represents whether the node is connected.

Return type bool

async send(*op*, *data=None*)
Send websocket payload to the node

Parameters

- **op** (*str*) – Operation name of the payload
- **data** (*Optional[dict]*) – Operation data to send with

Raises ***discodo.NodeNotConnected*** – The node is not connnected.

class discodo.**Nodes**(*iterable=()*, */*)
Represents a discodo node connection list.

You can also use it like `list`.

async connect()
Try to connect to all registered nodes.

Return type list

async getStatus()
Try to get status from all registered nodes.

Return type list

3.8.4 VoiceClient

class discodo.**VoiceClient**(*Node*, *id*, *guild_id*)
Represents a voice connection of the guild.

Variables

- ***Node*** (discodo.Node) – The node which the connection is connected with.
- **client** (discodo.DPYClient) – The client which the connection is binded.
- **loop** (*asyncio.AbstractEventLoop*) – The event loop that the client uses for operation.
- **id** (*str*) – The id of the voice client, which is used on restful api.
- **guild_id** (*int*) – The guild id which is connected to.
- **channel_id** (*Optional[int]*) – The channel id which is connected to.
- **dispatcher** (EventDispatcher) – The event dispatcher that the client dispatches events.
- **Queue** (*list*) – The queue of the guild, it is synced to node and readonly.

property autoplay
Represents the autoplay state of this guild.

Return type bool

property crossfade
Represents the crossfade duration of this guild.

Return type float

async destroy()
Destroy the client

async fetchContext()
Fetch the context from the node.

Return type dict

async fetchQueue(*ws=True*)
Fetch queue to force refresh the internal queue.

Parameters **ws** (*Optional[bool]*) – Whether to request queue on websocket or not.

Return type list[*AudioData* or *AudioSource*]

async fetchState()
Fetch current player state.

Return type dict

property filter
Represents the autoplay state of this guild.

Note: For more information, check the documentation of the FFmpeg Filter

Return type bool

async getCurrent()
Fetch current playing source

Return type *AudioSource*

async getOptions()
Get options of the player

Return type dict

async getSource(*query*)
Search the query and get source from extractor

Parameters **query** (*str*) – The query to search.

Return type *AudioData*

async getSubtitle(**args*, *callback*, ***kwargs*)
Request to send synced subtitle to discodo node and handle event to callback function.

`lang` or `url` is required.

Parameters

- **callback** (*callable*) – The callback function on subtitle event, must be coroutine function.
- **lang** (*Optional[str]*) – The language to get subtitle.
- **url** (*Optional[str]*) – The subtitle url to fetch.

Return type dict

async loadSource(*query*)
Search the query and put source to the queue

Parameters **query** (*str*) – The query to search.

Return type *AudioData* or list

async moveTo(*node*)
Move the player's current Node.

Parameters **node** (discodo.Node) – The node to move to.

async pause()
Pause the player

async putSource(*source*)
Search the query and get sources from extractor

Parameters **source** (*AudioData or AudioSource or list*) – The source to put on the queue.

Return type *AudioData* or *AudioSource* or list

async query(*op*, *data=None*, *event=None*, *timeout=10.0*)
Send websocket payload to the node with guild id and await response.

Parameters

- **op** (*str*) – Operation name of the payload
- **data** (*Optional[dict]*) – Operation data to send with
- **event** (*Optional[str]*) – Event name to receive response, defaults to `op`
- **timeout** (*Optional[float]*) – Seconds to wait for response

Raises

- **asyncio.TimeoutError** – The query is timed out.
- ***discodo.NodeException*** – The node returned some exceptions.

Return type Any

async requestSubtitle(*lang=None*, *url=None*)
Request to send synced subtitle to discodo node.

One of the parameters is required.

Parameters

- **lang** (*Optional[str]*) – The language to get subtitle.
- **url** (*Optional[str]*) – The subtitle url to fetch.

Return type dict

async resume()
Resume the player

async searchSources(*query*)
Search the query and get sources from extractor

Parameters **query** (*str*) – The query to search.

Return type list[*AudioData*]

async seek(*offset*)
Seek the player

Parameters **offset** (*float*) – The position to seek

async send(*op*, *data*)
Send websocket payload to the node with guild id

Parameters

- **op** (*str*) – Operation name of the payload
- **data** (*Optional[dict]*) – Operation data to send with

async setAutoplay(*autoplay*)
Set autoplay state of the player

Parameters **autoplay** (*bool*) – The autoplay state of the player to change.

Return type dict

async setContext(*data*)
Set the context to the node.

Parameters **data** (*dict*) – The context to set.

Return type dict

async setCrossfade(*crossfade*)
Set crossfade of the player

Parameters **crossfade** (*float*) – The crossfade of the player to change.

Return type dict

async setFilter(*filter*)
Set filter of the player

Parameters **crossfade** (*float*) – The filter object of the player to change.

Return type dict

async setOptions(***options*)
Set options of the player

Parameters

- **volume** (*Optional[float]*) – The volume of the player to change.
- **crossfade** (*Optional[float]*) – The crossfade of the player to change.
- **autoplay** (*Optional[bool]*) – The autoplay state of the player to change.
- **filter** (*Optional[dict]*) – The filter object of the player to change.

Return type dict

async setVolume(*volume*)
Set volume of the player

Parameters **volume** (*float*) – The volume of the player to change.

Return type dict

async shuffle()
Shuffle the queue

Return type list[*AudioData* or *AudioSource*]

async skip(*offset=1*)
Skip the source

Parameters **offset** (*int*) – how many to skip the sources

property volume
Represents the volume of this guild.

The maximum value is `2.0`

Note: This value is a percentage with decimal points. For example, `100%` is `1.0`, and `5%` is `0.05`.

Return type float

3.8.5 AudioData

class discodo.models.**AudioData**(*VoiceClient*, *data*)

Object with playback information

x == y

Checks if two AudioSources are equal.

x != y

Checks if two AudioSources are not equal.

Variables

- **tag** (*str*) – The tag of the object
- **id** (*str*) – The id of the source
- **title** (*Optional[str]*) – The title of the source
- **webpage_url** (*Optional[str]*) – The webpage url of the source
- **thumbnail** (*Optional[str]*) – The thumbnail url of the source
- **url** (*Optional[str]*) – The stream url of the source
- **duration** (*Optional[int]*) – The duration of the source
- **is_live** (*bool*) – Whether the source is live stream or not
- **uploader** (*Optional[str]*) – The uploader of the source
- **description** (*Optional[str]*) – The description of the source
- **subtitles** (*dict*) – The description of the source
- **chapters** (*dict*) – The description of the source
- **related** (*bool*) – Whether this source is added by autoplay
- **context** (*dict*) – The context of the object
- **start_position** (*float*) – The start position of the source

async getContext()

Get the context from the node.

Return type dict

async moveTo(*index*)

Move this source to index in the queue.

Parameters **index** (*int*) – The index to move to

Return type *AudioData*

async put()

Put the source into the queue.

Raises **ValueError** – The source is already in the queue.

Return type *AudioData*

async remove()

Remove this source from the queue.

Return type *AudioData*

async seek(*offset*)
Seek this source to offset.

Parameters **offset** (*float*) – The position to start playing

Return type *AudioData*

async setContext(*data*)
Set the context to the node.

Parameters **data** (*dict*) – The context to set.

Return type dict

3.8.6 AudioSource

class discodo.models.**AudioSource**(*VoiceClient*, *data*)
Object with playback information that is loaded

x == y
Checks if two AudioSources are equal.

x != y
Checks if two AudioSources are not equal.

Variables

- **tag** (*str*) – The tag of the object
- **id** (*str*) – The id of the source
- **title** (*Optional[str]*) – The title of the source
- **webpage_url** (*Optional[str]*) – The webpage url of the source
- **thumbnail** (*Optional[str]*) – The thumbnail url of the source
- **url** (*Optional[str]*) – The stream url of the source
- **duration** (*float*) – The duration of the source
- **is_live** (*bool*) – Whether the source is live stream or not
- **uploader** (*Optional[str]*) – The uploader of the source
- **description** (*Optional[str]*) – The description of the source
- **subtitles** (*dict*) – The description of the source
- **chapters** (*dict*) – The description of the source
- **related** (*bool*) – Whether this source is added by autoplay
- **context** (*dict*) – The context of the object
- **start_position** (*float*) – The start position of the source
- **seekable** (*bool*) – Whether the source is seekable or not
- **position** (*Optional[float]*) – The current position of the source

async getContext()
Get the context from the node.

Return type dict

async setContext(*data*)
Set the context to the node.

Parameters **data** (*dict*) – The context to set.

Return type dict

3.8.7 Errors

exception discodo.**DiscodoException**
The basic exception class of discodo

exception discodo.**EncryptModeNotReceived**
Exception that is thrown when trying to send packet before receveing encrypt mode.

It's only raise in DiscordVoiceClient

exception discodo.**NotPlaying**
Exception that is thrown when trying to operate something while not playing.

exception discodo.**VoiceClientNotFound**
Exception that is thrown when there is no voice client.

exception discodo.**NoSearchResults**
Exception that is thrown when there is no search results.

exception discodo.**OpusLoadError**
Exception that is thrown when loading libopus failed.

exception discodo.**HTTPException**(*status: int*, *data=None*)
Exception that is thrown when HTTP operation failed.

Variables

- ***status*** (*int*) – HTTP status code
- **description** (*str*) – Description of the HTTP status code
- **message** (*str*) – Server message with this request

exception discodo.**Forbidden**
Exception that is thrown when HTTP status code is 403.

exception discodo.**TooManyRequests**
Exception that is thrown when HTTP status code is 429.

exception discodo.**NotSeekable**
Exception that is thrown when trying to seek the source which is not seekable.

exception discodo.**NodeException**(*name*, *reason*)
Exception that is thrown when discodo node returns some exception.

exception discodo.**NodeNotConnected**
Exception that is thrown when there is no discodo node that is connected.

3.9 Event Reference

This section outlines the different types of events dispatched by discodo client.

Note: If you are using a standalone discodo node server while not using `DPYClient`, the events that you get will have something different. See this *Event Reference*.

To listen an event, use `EventDispatcher` of the `DPYClient`

```
import discord
import discodo

bot = discord.Client()
codo = discodo.DPYClient(bot)

# Using DPYClient.event

@codo.event("SOURCE_START")
async def start_event(VC, source):
    print(f"{VC} is now playing {source}")

# Using DPYClient.dispatcher.on

async def stop_event(VC, source):
    print(f"{VC} is stopped {source}")

codo.dispatcher.on("SOURCE_STOP", stop_event)
```

3.9.1 VC_CREATED(`VoiceClient`, `dict` data)

Called when the new voice client has successfully created. This is not the same as the client being fully connected.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
id	str	The id of the voice client

3.9.2 QUEUE_EVENT(`VoiceClient`, `dict` data)

Called when there is something changed in the queue of the voice client. If you are using `DPYClient`, Ignore this event.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
name	str	The name of the method
args	list	The arguments of the method

3.9.3 VC_CHANNEL_EDITED(`VoiceClient`, `dict` data)

Called when the voice channel of the voice client is changed.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
channel_id	int	The channel id of the voice client

3.9.4 putSource(`VoiceClient`, `dict` data)

Called when some sources are put in the queue.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
sources	list	The sources which is put

3.9.5 loadSource(`VoiceClient`, `dict` data)

Called when some sources are searched and put in the queue.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
source	Union[AudioData, list]	The sources which is searched and put

3.9.6 REQUIRE_NEXT_SOURCE(`VoiceClient`, `dict` data)

Called when the player needs next source to play. If you set `autoplay` as `True`, the related source will be put after this event.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
current	AudioSource	The source which the player is currently playing

3.9.7 SOURCE_START(`VoiceClient`, `dict` data)

Called when the player starts to play the source.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
source	AudioSource	The source which the player starts to play

3.9.8 SOURCE_STOP(`VoiceClient`, `dict` data)

Called when the player stops to play the source.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
source	AudioSource	The source which the player stops to play

3.9.9 PLAYER_TRACEBACK(`VoiceClient`, `dict` data)

Called when the player gets some traceback while trying to send packets to discord server.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
traceback	str	The traceback information which the player gets

3.9.10 SOURCE_TRACEBACK(`VoiceClient`, `dict` data)

Called when the player gets some traceback while trying to play the source. That source will be automatically removed from the queue.

Data Structure

Field	Type	Description
guild_id	int	The guild id of the voice client
source	Union[AudioData, AudioSource]	The source which the player gets traceback while trying to play
traceback	str	The traceback information which the player gets

PYTHON MODULE INDEX

d

discodo.utils.status, 34

HTTP ROUTING TABLE

/context

`GET /context`, 16
`POST /context`, 17

/current

`GET /current`, 24
`POST /current`, 25

/loadSource

`POST /loadSource`, 18

/options

`GET /options`, 18
`POST /options`, 19

/pause

`POST /pause`, 21

/putSource

`POST /putSource`, 17

/queue

`GET /queue`, 22
`GET /queue/{tag_or_index}`, 25
`POST /queue/{tag_or_index}`, 26
`DELETE /queue/{tag_or_index}`, 26

/resume

`POST /resume`, 21

/seek

`GET /seek`, 20
`POST /seek`, 20

/shuffle

`POST /shuffle`, 22

/skip

`POST /skip`, 21

INDEX

A

AudioData (*class in discodo.models*), 41
AudioSource (*class in discodo.models*), 42
autoplay() (*discodo.VoiceClient property*), 37

C

callback() (*discodo.utils.CallbackList static method*), 35
CallbackList (*class in discodo.utils*), 35
close() (*discodo.Node method*), 36
connect() (*discodo.DPYClient method*), 31
connect() (*discodo.Node method*), 36
connect() (*discodo.Nodes method*), 37
crossfade() (*discodo.VoiceClient property*), 37

D

destroy() (*discodo.DPYClient method*), 32
destroy() (*discodo.Node method*), 36
destroy() (*discodo.VoiceClient method*), 37
discodo.utils.status
    module, 34
DiscodoException, 43
disconnect() (*discodo.DPYClient method*), 32
discordDispatch() (*discodo.Node method*), 36
dispatch() (*discodo.utils.EventDispatcher method*), 33
DPYClient (*class in discodo*), 31

E

EncryptModeNotReceived, 43
event() (*discodo.DPYClient property*), 32
event() (*discodo.utils.EventDispatcher method*), 33
EventDispatcher (*class in discodo.utils*), 33

F

fetchContext() (*discodo.VoiceClient method*), 37
fetchQueue() (*discodo.VoiceClient method*), 37
fetchState() (*discodo.VoiceClient method*), 38
filter() (*discodo.VoiceClient property*), 38
Forbidden, 43

G

getBestNode() (*discodo.DPYClient method*), 32
getContext() (*discodo.models.AudioData method*), 41
getContext() (*discodo.models.AudioSource method*), 42
getCpuCount() (*in module discodo.utils.status*), 34
getCurrent() (*discodo.VoiceClient method*), 38
getFreePort() (*discodo.utils.tcp method*), 35
getMemory() (*in module discodo.utils.status*), 34
getNetworkInbound() (*in module discodo.utils.status*), 34
getNetworkOutbound() (*in module discodo.utils.status*), 34
getOptions() (*discodo.VoiceClient method*), 38
getProcessCpu() (*in module discodo.utils.status*), 34
getProcessMemory() (*in module discodo.utils.status*), 35
getProcessThreads() (*in module discodo.utils.status*), 35
getSource() (*discodo.VoiceClient method*), 38
getStatus() (*discodo.Node method*), 36
getStatus() (*discodo.Nodes method*), 37
getStatus() (*in module discodo.utils.status*), 35
getSubtitle() (*discodo.VoiceClient method*), 38
getTotalCpu() (*in module discodo.utils.status*), 35
getVC() (*discodo.DPYClient method*), 32
getVC() (*discodo.Node method*), 36
getWebsocket() (*discodo.DPYClient method*), 32

H

HTTPException, 43

I

is_connected() (*discodo.Node property*), 36

L

loadSource() (*discodo.VoiceClient method*), 38

M

module

discodo.utils.status, 34
moveTo() (*discodo.models.AudioData method*), 41
moveTo() (*discodo.VoiceClient method*), 38

N

Node (*class in discodo*), 35
NodeException, 43
NodeNotConnected, 43
Nodes (*class in discodo*), 37
NoSearchResults, 43
NotPlaying, 43
NotSeekable, 43

O

off() (*discodo.utils.EventDispatcher method*), 33
offAny() (*discodo.utils.EventDispatcher method*), 34
on() (*discodo.utils.EventDispatcher method*), 34
onAny() (*discodo.utils.EventDispatcher method*), 34
OpusLoadError, 43

P

pause() (*discodo.VoiceClient method*), 38
put() (*discodo.models.AudioData method*), 41
putSource() (*discodo.VoiceClient method*), 38

Q

query() (*discodo.VoiceClient method*), 39

R

registerNode() (*discodo.DPYClient method*), 33
remove() (*discodo.models.AudioData method*), 41
requestSubtitle() (*discodo.VoiceClient method*), 39
resume() (*discodo.VoiceClient method*), 39

S

searchSources() (*discodo.VoiceClient method*), 39
seek() (*discodo.models.AudioData method*), 41
seek() (*discodo.VoiceClient method*), 39
send() (*discodo.Node method*), 36
send() (*discodo.VoiceClient method*), 39
setAutoplay() (*discodo.VoiceClient method*), 39
setContext() (*discodo.models.AudioData method*), 42
setContext() (*discodo.models.AudioSource method*), 42
setContext() (*discodo.VoiceClient method*), 39
setCrossfade() (*discodo.VoiceClient method*), 40
setFilter() (*discodo.VoiceClient method*), 40
setOptions() (*discodo.VoiceClient method*), 40
setVolume() (*discodo.VoiceClient method*), 40
shuffle() (*discodo.VoiceClient method*), 40
skip() (*discodo.VoiceClient method*), 40

T

TooManyRequests, 43

U

URL() (*discodo.Node property*), 36

V

VoiceClient (*class in discodo*), 37
VoiceClientNotFound, 43
voiceClients() (*discodo.DPYClient property*), 33
volume() (*discodo.VoiceClient property*), 40

W

wait_for() (*discodo.utils.EventDispatcher method*), 34
WS_URL() (*discodo.Node property*), 36