Useless-Lavalink Documentation

Release 1.4.6

tekulvw Cog-Creators jotonedev

Mar 13, 2022

CONTENTS:

A Lavalink client library written for Python 3.9 using the AsyncIO framework. This is a fork of Red-Lavalink.

To install:

```
pip install useless_lavalink
```

CHAPTER

ONE

USAGE

```
import lavalink
from discord.ext.commands import Bot

bot = Bot()


@bot.event
async def on_ready():
    await lavalink.initialize(bot)
    await lavalink.add_node(
        bot, host='localhost', password='password',  ws_port=2333
    )


async def search_and_play(voice_channel, search_terms):
    player = await lavalink.connect(voice_channel)
    tracks = await player.search_yt(search_terms)
    player.add(tracks[0])
    await player.play()
```

CHAPTER

TWO

SHUFFLING

```
def shuffle_queue(player_id, forced=True):
    player = lavalink.get_player(player_id)
    if not forced:
        player.maybe_shuffle(sticky_songs=0)
        """
        `player.maybe_shuffle` respects `player.shuffle`
        And will only shuffle if `player.shuffle` is True.

        `player.maybe_shuffle` should be called every time
        you would expect the queue to be shuffled.

        `sticky_songs=0` will shuffle every song in the queue.
        """
    else:
        player.force_shuffle(sticky_songs=3)
        """
        `player.force_shuffle` does not respect `player.shuffle`
        And will always shuffle the queue.

        `sticky_songs=3` will shuffle every song after the first 3 songs in the queue.
        """
```

When shutting down, be sure to do the following:

```
await lavalink.close(bot)
```

2.1 API Reference

2.1.1 Main functions

await lavalink.**initialize**(*bot: discord.ext.commands.bot.Bot*)
Setup event and update listener

Important: This function must only be called AFTER the bot has received its "on_ready" event!

Parameters **bot** (*discord.ext.commands.Bot*) – An instance of a discord *Bot* object.

await lavalink.**add_node**(*bot: discord.ext.commands.bot.Bot*, *host: str*, *password: str*, *ws_port: int*, *timeout: int = 30*, *resume_key: Optional[str] = None*, *resume_timeout: int = 60*)

Create and initialize a new node

Important: This function must only be called AFTER the initialize function

Parameters

- **bot** (*discord.ext.commands.Bot*) – An instance of a discord *Bot* object.
- **host** (*str*) – The hostname or IP address of the Lavalink node.
- **password** (*str*) – The password of the Lavalink node.
- **ws_port** (*int*) – The websocket port on the Lavalink Node.
- **timeout** (*int*) – Amount of time to allow retries to occur, `None` is considered forever.
- **resume_key** (*Optional[str]*) – A resume key used for resuming a session upon re-establishing a WebSocket connection to Lavalink.
- **resume_timeout** (*int*) – How long the node should wait for a connection while disconnected before clearing all players.

await lavalink.**connect**(*channel: discord.channel.VoiceChannel*, *deafen: bool = False*)

Connects to a discord voice channel.

This is the publicly exposed way to connect to a discord voice channel. The *initialize()* function must be called first!

Parameters

- **deafen** (*bool*) – Prevent the bot from listening others
- **channel** (*discord.VoiceChannel*) – The channel to move to

Returns The created Player object.

Return type *Player*

Raises **IndexError** – If there are no available lavalink nodes ready to connect to discord.

lavalink.**get_player**(*guild_id: int*) → *lavalink.player.Player*

Get the player of a guild

Parameters **guild_id** (*int*) – The guild id

Returns The player of the given guild id

Return type *Player*

await lavalink.**close**(*bot: discord.ext.commands.bot.Bot*)

Closes the lavalink connection completely.

Parameters **bot** (*discord.ext.commands.Bot*) –

lavalink.**register_event_listener**(*coro: Callable[[...], Awaitable[None]]*)

Registers a coroutine to receive lavalink event information.

This coroutine will accept three arguments: *Player*, *LavalinkEvents*, and possibly an extra. The value of the extra depends on the value of the second argument.

If the second argument is *LavalinkEvents.TRACK_END*, the extra will be a *TrackEndReason*.

If the second argument is *LavalinkEvents.TRACK_EXCEPTION*, the extra will be a dictionary with `message`, `cause`, and `severity` keys.

If the second argument is *LavalinkEvents.TRACK_STUCK*, the extra will be the threshold milliseconds that the track has been stuck for.

If the second argument is *LavalinkEvents.TRACK_START*, the extra will be a track identifier string.

If the second argument is any other value, the third argument will not exist.

Parameters **coro** (coroutine) – A coroutine function that accepts the arguments listed above.

Raises **TypeError** – If `coro` is not a coroutine.

lavalink.**register_update_listener**(*coro: Callable[[...], Awaitable[None]]*)
Registers a coroutine to receive lavalink player update information.

This coroutine will accept two arguments: an instance of *Player* and an instance of *PlayerState*.

Parameters **coro** (coroutine) –

Raises **TypeError** – If `coro` is not a coroutine.

lavalink.**register_stats_listener**(*coro: Callable[[...], Awaitable[None]]*)
Registers a coroutine to receive lavalink server stats information.

This coroutine will accept a single argument which will be an instance of *Stats*.

Parameters **coro** (coroutine) –

Raises **TypeError** – If `coro` is not a coroutine.

lavalink.**unregister_event_listener**(*coro: Callable[[...], Awaitable[None]]*)
Unregisters coroutines from being event listeners.

Parameters **coro** (coroutine) –

lavalink.**unregister_update_listener**(*coro: Callable[[...], Awaitable[None]]*)
Unregisters coroutines from being player update listeners.

Parameters **coro** (coroutine) –

lavalink.**unregister_stats_listener**(*coro: Callable[[...], Awaitable[None]]*)
Unregisters coroutines from being server stats listeners.

Parameters **coro** (coroutine) –

lavalink.**all_players**() → Tuple[*lavalink.player.Player*]
Get all the players

lavalink.**all_connected_players**() → Tuple[*lavalink.player.Player*]
Get all the connected players

lavalink.**active_players**() → Tuple[*lavalink.player.Player*]
Get all the active players

lavalink.utils.**format_time**(*time*)
Formats the given time into HH:MM:SS

2.1.2 Enums

class lavalink.**LavalinkEvents**(*value*)
An enumeration of the Lavalink Track Events.

TRACK_END = 'TrackEndEvent'
The track playback has ended.

TRACK_EXCEPTION = 'TrackExceptionEvent'
There was an exception during track playback.

TRACK_STUCK = 'TrackStuckEvent'
Track playback got stuck during playback.

TRACK_START = 'TrackStartEvent'
The track playback started.

WEBSOCKET_CLOSED = 'WebSocketClosedEvent'
Websocket has been closed.

FORCED_DISCONNECT = 'ForcefulDisconnectEvent'
Connection has been disconnect, do not attempt to reconnect.

QUEUE_END = 'QueueEndEvent'
This is a custom event generated by this library to denote the end of all tracks in the queue.

class lavalink.**TrackEndReason**(*value*)
The reasons why track playback has ended.

FINISHED = 'FINISHED'
The track reached the end, or the track itself ended with an exception.

LOAD_FAILED = 'LOAD_FAILED'
The track failed to start, throwing an exception before providing any audio.

STOPPED = 'STOPPED'
The track was stopped due to the player being stopped.

REPLACED = 'REPLACED'
The track stopped playing because a new track started playing.

CLEANUP = 'CLEANUP'
The track was stopped because the cleanup threshold for the audio player was reached.

class lavalink.**FiltersOp**(*value*)
An enumeration of all filters available to use

VOLUME = 'volume'
Manage song volume

EQUALIZER = 'equalizer'
Edit equalizer

KARAOKE = 'karaoke'
Remove vocals

TIMESCALE = 'timescale'
Changes the speed, pitch, and rate

TREMOLO = 'tremolo'
Create a shuddering effect

VIBRATO = 'vibrato'
Produce a pulsating change of the pitch

ROTATION = 'rotation'
Audio Panning

DISTORTION = 'distortion'
Distortion effect

CHANNEL_MIX = 'channelMix'
Mixes both channels (left and right)

LOW_PASS = 'lowPass'
Higher frequencies get suppressed, while lower frequencies pass through this filter

class lavalink.**NodeState**(*value*)
An enumeration.

CONNECTING = 0
The client is connecting to the node

READY = 1
The node is ready to send and receive requests

RECONNECTING = 2
The connection with the node was lost. Now it's reconnecting with it

DISCONNECTING = 3
Closing connection with the node

class lavalink.**PlayerState**(*value*)
An enumeration.

CREATED = -1
Player was created

CONNECTING = 0
The client is connecting to the player

READY = 1
The player is ready to reproduce

NODE_BUSY = 2
The node cannot respond to the player requests

RECONNECTING = 3
The connection with the player was lost. Now it's reconnecting with it

DISCONNECTING = 4
Closing connection with the player

2.1.3 Player

class lavalink.**Player**(*client: Bot = None*, *channel: discord.VoiceChannel = None*, *node:* Node *= None*, *ssl: bool = False*)
The Player class represents the current state of playback. It also is used to control playback and queue tracks.

The existence of this object guarantees that the bot is connected to a voice channel.

channel
The channel the bot is connected to.

Type discord.VoiceChannel

queue
list of Track

Type deque[*Track*]

position
The seeked position in the track of the current playback.

Type int

current
The current playing track

Type *Track*

repeat
Repeat the current track

Type bool

loop_queue
Repeat the current queue

Type bool

shuffle
The newly added tracks to the queue gets shuffled

Type bool

property is_auto_playing: bool
Current status of playback

property is_playing: bool
Current status of playback

property paused: bool
Player's paused state.

property volume: int
The current volume.

property ready: bool
Whether the underlying node is ready for requests.

property connected: bool
Whether the player is ready to be used.

await on_voice_server_update(*data: dict[str, Any]*)
Send voice server update data to the node

Parameters **data** (*dict[str, Any]*) – The data to send

await on_voice_state_update(*data: dict*)
Send voice state update data to the node

Parameters **data** (*dict[str, Any]*) – The data to send

await wait_until_ready(*timeout: Optional[float] = None*, *no_raise: bool = False*) → bool
Waits for the underlying node to become ready.

Parameters

- **timeout** (*Optional[float]*) – The time to wait for the node to get ready. A timeout of None means to wait indefinitely.

- **no_raise** (*bool*) – If no_raise is set, returns false when a timeout occurs instead of propagating TimeoutError.

await connect(*timeout: float = 2.0*, *reconnect: bool = False*, *deafen: bool = False*)
Connects to the voice channel associated with this Player.

Parameters

- **deafen** (*bool*) – Prevent the bot from listening others'
- **timeout** (*float*) – Actually is not used, but must be present to match the signature
- **reconnect** (*bool*) – Actually is not used, but must be present to match the signature

await move_to(*channel: discord.channel.VoiceChannel*, *deafen: bool = False*)
Moves this player to a voice channel.

Parameters

- **deafen** (*bool*) – Prevent the bot from listening others
- **channel** (*discord.VoiceChannel*) – The channel to move to

Raises **TypeError** – If the channel is in another guild

await disconnect(*force: bool = False*)
Disconnects this player from its voice channel.

Parameters **force** (*bool*) – Force the player to disconnect

store(*key: Union[str, int]*, *value: Any*)
Stores a metadata value by key.

Parameters

- **key** (*Union[str, int]*) –
- **value** (*Any*) –

fetch(*key: Union[str, int]*, *default: Optional[Any] = None*) → Any
Returns a stored metadata value.

Parameters

- **key** – Used to store metadata.
- **default** – Optional, used if the key doesn't exist.

Returns If the key doesn't exist returns the default value, otherwise the set value

Return type Any

await update_state(*state:* lavalink.enums.PlayerState)
Change the current player state :param state: The state to set :type state: PlayerState

await handle_event(*event:* lavalink.enums.LavalinkEvents, *extra: Any*)
Handles various Lavalink Events.

If the event is TRACK END, extra will be TrackEndReason.

If the event is TRACK EXCEPTION, extra will be the string reason.

If the event is TRACK STUCK, extra will be the threshold ms.

Parameters

- **event** (*node.LavalinkEvents*) –
- **extra** (*Any*) –

await handle_player_update(*state:* lavalink.tuples.PositionTime)
Handles player updates from lavalink.

Parameters **state** (*websocket.PlayerState*) –

add(*requester: discord.user.User*, *track:* lavalink.rest_api.Track)
Adds a track to the queue.

Parameters

- **requester** (*discord.User*) – Who requested the track.
- **track** (Track) – Result from any of the lavalink track search methods.

maybe_shuffle(*sticky_songs: int = 1*)
Shuffle

force_shuffle(*sticky_songs: int = 1*)
Shuffle the queue

await play()
Starts playback from lavalink.

await stop()
Stops playback from lavalink.

Important: This method will clear the queue.

await skip()
Skips to the next song.

await pause(*pause: bool = True*, *timed: Optional[int] = None*)
Pauses the current song.

Parameters

- **pause** (*bool*) – Set to `False` to resume.
- **timed** (*Optional[int]*) – If an int is given the op will be called after it.

await set_volume(*volume: int*)
Sets the volume of Lavalink.

Parameters **volume** (*int*) – Between 0 and 150

await seek(*position: int*)
If the track allows it, seeks to a position.

Parameters **position** (*int*) – Between 0 and track length.

await bass_boost()
Bass boost the song

await nightcore()
Apply the effect nightcore on the song

await random_distortion()
Apply a random distortion

Warning: It can be extremely painful to listen to

await reset_filter()
Remove any applied filter

await equalizer(*band: int*, *gain: float = 0.25*)
Change the equalizer

Parameters

- **band** (*int*) – 0 x 14
- **gain** (*float*) – is the multiplier for the given band -0.25 x 1

cleanup()
This method *must* be called to ensure proper clean-up during a disconnect.

It is advisable to call this from within *disconnect()* when you are completely done with the voice protocol instance.

This method removes it from the internal state cache that keeps track of currently alive voice clients. Failure to clean-up will cause subsequent connections to report that it's still connected.

await get_tracks(*query: str*) → Tuple[*lavalink.rest_api.Track*, ...]
Gets tracks from lavalink.

Parameters **query** (*str*) –

Return type Tuple[*Track*, . . .]

await karaoke(*level: float = 1.0*, *mono_level: float = 1.0*, *filter_band: float = 220.0*, *filter_width: float = 100.0*)
Remove the vocals

Parameters

- **level** (*float*) – how much to filter
- **mono_level** (*float*) – how much to filter
- **filter_band** (*float*) – the frequency band to filter
- **filter_width** (*float*) – the frequency width to filter

await load_tracks(*query: str*) → *lavalink.rest_api.LoadResult*
Executes a loadtracks request. Only works on Lavalink V3.

Parameters **query** (*str*) –

Return type *LoadResult*

await search_sc(*query: str*) → *lavalink.rest_api.LoadResult*
Gets track results from SoundCloud from Lavalink.

Parameters **query** (*str*) –

Return type list of Track

await search_yt(*query: str*) → *lavalink.rest_api.LoadResult*
Gets track results from YouTube from Lavalink.

Parameters **query** (*str*) –

Return type list of Track

await rotation(*rotation: Optional[int] = None*)
Rotates the sound around the stereo channels/user headphones Audio Panning

Parameters **rotation** (*Optional[int]*) – The frequency of the audio rotating around the listener in Hz

await timescale(*speed: float = 1.0*, *pitch: float = 1.0*, *rate: float = 1.0*)
Changes the speed, pitch, and rate

Parameters

- **speed** (*float*) – Should be >= 0
- **pitch** (*float*) – Should be >= 0
- **rate** (*float*) – Should be >= 0

await vibrato(*frequency: float = 2.0*, *depth: float = 0.5*)
Oscillates the pitch.

Parameters

- **frequency** (*float*) – 0 < x 14
- **depth** (*float*) – 0 < x 1

await tremolo(*frequency: float = 2.0*, *depth: float = 0.5*)
Uses amplification to create a shuddering effect. Oscillates the volume

Parameters

- **frequency** (*float*) – Should be >= 0
- **depth** (*float*) – 0 < x 1

await distortion(*sin_offset: float = 0*, *sin_scale: float = 1*, *cos_offset: float = 0*, *cos_scale: float = 1*, *tan_offset: float = 0*, *tan_scale: float = 1*, *offset: float = 0*, *scale: float = 1*)
Distortion effect

Parameters

- **sin_offset** (*int*) –
- **sin_scale** (*int*) –
- **cos_offset** (*float*) –
- **cos_scale** (*float*) –
- **tan_offset** (*float*) –
- **tan_scale** (*float*) –
- **offset** (*float*) –
- **scale** (*float*) –

await channel_mix(*left_to_left: float = 1.0*, *left_to_right: float = 0.0*, *right_to_left: float = 0.0*, *right_to_right: float = 1.0*)
Mixes both channels (left and right)

Parameters

- **left_to_left** (*float*) –
- **left_to_right** (*float*) –
- **right_to_left** (*float*) –
- **right_to_right** (*float*) –

await low_pass(*smoothing: float = 20.0*)
Higher frequencies get suppressed, while lower frequencies pass through this filter

Parameters **smoothing** (*float*) – how much to suppress

2.1.4 Node

class lavalink.**Node**(*_loop: asyncio.base_events.BaseEventLoop, event_handler: Callable, host: str, password: str, port: int, user_id: int, num_shards: int, resume_key: Optional[str] = None, resume_timeout: int = 60, bot: Optional[discord.ext.commands.bot.Bot] = None*)

await connect(*timeout: Optional[int] = None, ssl: bool = False*)
Connects to the Lavalink player event websocket.

Parameters

- **ssl** (*bool*) – Whether to use the *wss://* protocol.
- **timeout** (*int*) – Time after which to timeout on attempting to connect to the Lavalink websocket, None is considered never, but the underlying code may stop trying past a certain point.

Raises **asyncio.TimeoutError** – If the websocket failed to connect after the given time.

property lavalink_major_version: str
Get the major version of the lavalink node

property ready: bool
Whether the underlying node is ready for requests.

await listener()
Listener task for receiving ops from Lavalink.

await create_player(*channel: Union[discord.channel.VoiceChannel, discord.channel.StageChannel], deafen: bool = False*) → *lavalink.player.Player*
Connects to a discord voice channel.

This function is safe to repeatedly call as it will return an existing player if there is one.

Parameters

- **deafen** –
- **channel** –

Returns The created Player object.

Return type *Player*

get_player(*guild_id: int*) → *lavalink.player.Player*
Gets a Player object from a guild ID.

Parameters **guild_id** (*int*) – Discord guild ID.

Return type *Player*

Raises **KeyError** – If that guild does not have a Player, e.g. is not connected to any voice channel.

add_player(*guild_id: int, player:* lavalink.player.Player)
Register a player

remove_player(*player:* lavalink.player.Player)
Remove a player

await disconnect()
Shuts down and disconnects the websocket.

await destroy_guild(*guild_id: int*)
Disconnect from a guild and delete the player

Parameters **guild_id** (*int*) –

await stop(*guild_id: int*)
Stop the player on the given guild id

Parameters **guild_id** (*int*) –

await play(*guild_id: int*, *track:* lavalink.rest_api.Track, *replace: bool = True*, *start: int = 0*, *pause: bool = False*)
Start playing on the given guild id

Parameters

- **guild_id** (*int*) –
- **track** (Track) – The track to play
- **replace** (*bool*) – If set to true, stop the current song and play the given one
- **start** (*int*) – The number of milliseconds to offset the track by
- **pause** (*bool*) – Pause the current track

await pause(*guild_id: int*, *paused: bool*)
Pause the current track

Parameters

- **guild_id** (*int*) –
- **paused** (*bool*) – If set to True pause the track, otherwise unpause it

await volume(*guild_id: int*, *_volume: int*)
Set the volume of the track :param guild_id: :type guild_id: int :param _volume: Volume may range from 0 to 1000 :type _volume: int

await seek(*guild_id: int*, *position: int*)
Seek the current track

Parameters

- **guild_id** (*int*) –
- **position** (*int*) – The position is in milliseconds.

await equalizer(*guild_id: int*, *bands: list[*lavalink.tuples.EqualizerBands*]*)
Set the equalizer

Parameters

- **guild_id** (*int*) –
- **bands** (*list[*EqualizerBands*]*) – A list of bands to change

await karaoke(*guild_id: int*, *level: float = 1.0*, *mono_level: float = 1.0*, *filter_band: float = 220.0*, *filter_width: float = 100.0*)
Uses equalization to eliminate part of a band, usually targeting vocals.

Parameters

- **guild_id** (*int*) –

- **level** (*float*) – how much to filter
- **mono_level** (*float*) – how much to filter
- **filter_band** (*float*) – the frequency band to filter
- **filter_width** (*float*) – the frequency width to filter

await time_scale(*guild_id: int*, *speed: float = 1.0*, *pitch: float = 1.0*, *rate: float = 1.0*)
Changes the speed, pitch, and rate. All default to 1.

Parameters

- **guild_id** (*int*) –
- **speed** (*float*) – Should be >= 0
- **pitch** (*float*) – Should be >= 0
- **rate** (*float*) – Should be >= 0

await tremolo(*guild_id: int*, *frequency: float = 2.0*, *depth: float = 0.5*)
Uses amplification to create a shuddering effect, where the volume quickly oscillates.

Parameters

- **guild_id** (*int*) –
- **frequency** (*float*) – Should be >= 0
- **depth** (*float*) – 0 < x 1

await vibrato(*guild_id: int*, *frequency: float = 2.0*, *depth: float = 0.5*)
Similar to tremolo. While tremolo oscillates the volume, vibrato oscillates the pitch.

Parameters

- **guild_id** (*int*) –
- **frequency** (*float*) – 0 < x 14
- **depth** (*float*) – 0 < x 1

await rotation(*guild_id: int*, *rotation: int = 0*)
Rotates the sound around the stereo channels/user headphones aka Audio Panning. It can produce an effect similar to: https://youtu.be/QB9EB8mTKcc (without the reverb)

Parameters

- **guild_id** (*int*) –
- **rotation** (*Optional[int]*) – The frequency of the audio rotating around the listener in Hz

await distortion(*guild_id: int*, *sin_offset: int = 0*, *sin_scale: int = 1*, *cos_offset: int = 0*, *cos_scale: int = 1*, *tan_offset: int = 0*, *tan_scale: int = 1*, *offset: int = 0*, *scale: int = 1*)
Distortion effect

Parameters

- **guild_id** (*int*) –
- **sin_offset** (*int*) –
- **sin_scale** (*int*) –
- **cos_offset** (*float*) –
- **cos_scale** (*float*) –

- **tan_offset** (*float*) –
- **tan_scale** (*float*) –
- **offset** (*float*) –
- **scale** (*float*) –

await channel_mix(*guild_id: int*, *left_to_left: float = 1.0*, *left_to_right: float = 0.0*, *right_to_left: float = 0.0*, *right_to_right: float = 1.0*)

Mixes both channels (left and right), with a configurable factor on how much each channel affects the other. With the defaults, both channels are kept independent from each other. Setting all factors to 0.5 means both channels get the same audio. :param guild_id: :type guild_id: int :param left_to_left: :type left_to_left: float :param left_to_right: :type left_to_right: float :param right_to_left: :type right_to_left: float :param right_to_right: :type right_to_right: float

await low_pass(*guild_id: int*, *smoothing: float = 0.0*)

Higher frequencies get suppressed, while lower frequencies pass through this filter, thus the name low pass.

Parameters

- **guild_id** (*int*) –
- **smoothing** (*float*) – how much to suppress

await reset_filter(*guild_id: int*)

Remove any applied filter

Parameters **guild_id** (*int*) –

class lavalink.**NodeStats**(*data: dict[str, Any]*)

The NodeStats class contains the performance stats of a certain node

uptime

Type int

players

Type int

playing_players

Type int

memory_free

Type int

memory_used

Type int

memory_allocated

Type int

memory_reservable

Type int

cpu_cores

Type int

system_load

Type float

lavalink_load

Type float

frames_sent

Type int

frames_nulled

Type int

frames_deficit

Type int

class lavalink.**Stats**(*memory: dict[str, int]*, *players: int*, *active_players: int*, *cpu: dict[str, Union[str, float]]*, *uptime: int*)

The NodeStats class contains the performance stats of a certain node

uptime

Type int

players

Type int

active_players

Type int

memory

Type *MemoryInfo*

cpu_info

Type *CPUInfo*

2.1.5 Rest API

class lavalink.rest_api.**Track**(*data: dict[str, Any]*)
Information about a Lavalink track.

requester
The user who requested the track.

Type discord.User

track_identifier
Track identifier used by the Lavalink player to play tracks.

Type str

identifier
Track identifier on YouTube

Type str

seekable
Boolean determining if seeking can be done on this track.

Type bool

author
The author of this track.

Type str

length
The length of this track in milliseconds.

Type int

is_stream
Determines whether Lavalink will stream this track.

Type bool

position
Current seeked position to begin playback.

Type int

title
Title of this track.

Type str

uri
The playback url of this track.

Type str

start_timestamp
The track start time in milliseconds as provided by the query.

Type int

property thumbnail: Optional[str]
Returns a thumbnail URL for YouTube tracks.

class lavalink.rest_api.**RESTClient**(*player:* Player, *ssl: bool = False*)
Client class used to access the REST endpoints on a Lavalink node.

player
The player to use

Type *Player*

node
The node used by the player

Type *Node*

state
The current player state

Type *PlayerState*

await load_tracks(*query: str*) → *lavalink.rest_api.LoadResult*
Executes a loadtracks request. Only works on Lavalink V3.

Parameters **query** (*str*) –

Return type *LoadResult*

await get_tracks(*query: str*) → Tuple[*lavalink.rest_api.Track*, ...]
Gets tracks from lavalink.

Parameters **query** (*str*) –

Return type Tuple[*Track*, ...]

await search_yt(*query: str*) → *lavalink.rest_api.LoadResult*
Gets track results from YouTube from Lavalink.

Parameters **query** (*str*) –

Return type list of Track

await search_sc(*query: str*) → *lavalink.rest_api.LoadResult*
Gets track results from SoundCloud from Lavalink.

Parameters **query** (*str*) –

Return type list of Track

class lavalink.rest_api.**LoadResult**(*data: dict[str, Any]*)
The result of a load_tracks request.

load_type
The result of the loadtracks request

Type LoadType

playlist_info
The playlist information detected by Lavalink

Type Optional[*PlaylistInfo*]

tracks
The tracks that were loaded, if any

Type tuple[*Track*, ...]

property exception_message: Optional[str]
On Lavalink V3, if there was an exception during a load or get tracks call this property will be populated with the error message. If there was no error this property will be `None`.

2.1.6 Tuples

class lavalink.tuples.**PositionTime**(*position*, *time*, *connected*)

position: int
Alias for field number 0

time: int
Alias for field number 1

connected: bool
Alias for field number 2

class lavalink.tuples.**MemoryInfo**(*reservable*, *used*, *free*, *allocated*)

reservable: int
Alias for field number 0

used: int
Alias for field number 1

free: int
Alias for field number 2

allocated: int
Alias for field number 3

class lavalink.tuples.**CPUInfo**(*cores*, *systemLoad*, *lavalinkLoad*)

cores: int
Alias for field number 0

systemLoad: float
Alias for field number 1

lavalinkLoad: float
Alias for field number 2

class lavalink.tuples.**EqualizerBands**(*band*, *gain*)

band: int
Alias for field number 0

gain: float
Alias for field number 1

class lavalink.tuples.**PlaylistInfo**(*name*, *selectedTrack*)

name: str
Alias for field number 0

selectedTrack: int
Alias for field number 1

CHAPTER

THREE

SITEMAP

- genindex

PYTHON MODULE INDEX

INDEX

I

K

L

M

N

O

P

Q

R

S

T

U

V

W